

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE**

DALTON DANTAS DE OLIVEIRA

**PROJETO PARA INCORPORAÇÃO DE SUPORTE A TESTES FUNCIONAIS
DE SOFTWARE À FERRAMENTA DE GERENCIAMENTO IPOJECT**

*Orientador: Ricardo Wendell Rodrigues da
Silveira, MSc.*

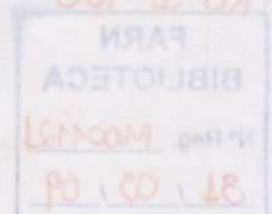
DALTON DANTAS DE OLIVEIRA

**PROJETO PARA INCORPORAÇÃO DE SUPORTE A TESTES FUNCIONAIS
DE SOFTWARE À FERRAMENTA DE GERENCIAMENTO IPOJECT**

Monografia apresentada ao Curso de Especialização em Desenvolvimento de Sistemas Corporativos para Web da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte como requisito obrigatório para obtenção do grau de Especialista em Desenvolvimento de Sistemas Corporativos.

Orientador: Ricardo Wendell Rodrigues da Silveira, MSc.

Natal
2008



Catálogo na Publicação – Biblioteca da FARN
Setor de Processos Técnicos

Oliveira, Dalton Dantas de

Projeto para incorporação de suporte a testes funcionais de software à ferramenta de gerenciamento IPROJECT / Dalton Dantas de Oliveira. – Natal: FARN, 2008.

59f.

Orientador: Ricardo Wendell Rodrigues da Silveira (Msc.)

Monografia (Especialização em Desenvolvimento de Sistemas Corporativos) – Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte.

1. Desenvolvimento de Software – Monografia. 2. Programas de Computadores – Monografia. 3. Testes Funcionais – Monografia. I. Título.

RN/FARN/BC

CDU 004.43



004.43
046p
2008
2x.01

DALTON DANTAS DE OLIVEIRA

Agradeco a Deus por tudo, pois sem ele nada seria possivel.

A minha esposa Anaik, e ao meu filho Dante que sempre me apoiam e apoia dos
momentos de ausencias.

A meus pais e meu irmão que sempre me incentivaram a superar novos desafios.

A Edson Alvarado, Joilson Abrantes, Idalmécio Dantas e a todos da equipe de testes

**PROJETO PARA INCORPORAÇÃO DE SUPORTE A TESTES FUNCIONAIS
DE SOFTWARE À FERRAMENTA DE GERENCIAMENTO IPOJECT**

A Gleydson Lima, Camilo de Lelis, Edipo Elder, Diego Medeiros, Victor Hugo,
Leandro Campos, Fábio Alexandre, Raphaela Galhardo, Agostinho Campos e todos os
amigos da SINFO que me ajudaram em mais essa jornada.

Um agradecimento especial ao amigo e orientador Ricardo Wendell pela importante

Monografia apresentada ao Curso de Desenvolvimento de Sistemas Corporativos da
Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte, aprovada em _____
de _____ de _____, pela banca examinadora composta pelos seguintes membros:



Ricardo Wendell Rodrigues da Silveira, MSc.
Orientador

Gleydson de Azevedo Ferreira Lima, MSc.
Membro Examinador

AGRADECIMENTOS

Agradeço a Deus por tudo, pois sem ele nada seria possível.

A minha esposa Anauk, e ao meu filho Dante que sempre me apoiaram apesar dos momentos de ausências.

A meus pais e meu irmão que sempre me incentivaram a superar novos desafios.

A Edson Alyppyo, Joilson Abrantes, Idelmárcia Dantas e a todos da equipe de testes e requisitos da SINFO pelas valiosas contribuições.

A Gleydson Lima, Camilo de Lélis, Édipo Elder, Diogo Medeiros, Victor Hugo, Leonardo Campos, Fábio Alexandre, Raphaela Galhardo, Agostinho Campos e todos os amigos da SINFO que me ajudaram em mais essa jornada.

Um agradecimento especial ao amigo e orientador Ricardo Wendell pela importante colaboração e paciência.

E a todas as pessoas que convivem comigo, pela oportunidade de conhecê-los.

Posso reclamar porque está chovendo ou agradecer as águas por lavarem a poluição. Posso ficar triste por não ter dinheiro ou me sentir encorajado para administrar minhas finanças, evitando a desperdição. Posso reclamar sobre minha saúde ou dar graças por estar vivo.

Posso me queixar dos meus pais por não terem me dado tudo o que eu queria ou posso ser grato por ter nascido. Posso reclamar por ter que trabalhar ou agradecer por ter trabalho. Posso sentir tédio com o trabalho doméstico ou agradecer a Deus por ter um teto para morar.

Posso lembrar decepções com amigos ou me entusiasmar com a possibilidade de fazer novas amizades. Se as coisas não saírem como planejei posso ficar feliz por ter hoje para reconsiderar.

O dia está na minha frente esperando para ser o que eu quiser. E aqui estou eu, o escultor que pode dar forma.

Tudo depende só de mim.

Charles Chaplin

RESUMO

Este trabalho pretende mostrar a importância dos testes num processo de desenvolvimento de software, fazendo uma descrição sobre a atividade de testes e os diversos tipos de testes, dando ênfase principalmente aos testes funcionais. Serão apresentadas diversas técnicas de testes funcionais e como estas técnicas são utilizadas na prática nos sistemas desenvolvidos pela Sinfon/PRN -- SIPAC, SIGAA e SIGRH. Em seguida será analisado o uso de uma ferramenta de gerenciamento de software, o IProject, mostrando seu projeto inicial, como ele pode auxiliar os gerentes na administração do fluxo das tarefas e como pode auxiliar o processo de desenvolvimento. Será analisado também o aprimoramento do projeto ao adicionar a gestão das atividades de testes à ferramenta IProject, abordando as tecnologias usadas, voltadas ao desenvolvimento de sistemas web orientado a objetos, as operações realizadas no projeto inicial do IProject, os novos casos de uso que surgiram com o aprimoramento da nova funcionalidade e por fim, uma análise dos resultados obtidos pelos testes.

Palavras-chave: Testes, Testes funcionais, Processo de desenvolvimento, Ferramenta de gerenciamento de software.

Só depende de nós ...

"Hoje levantei cedo pensando no que tenho a fazer antes que o relógio marque meia noite. É minha função escolher que tipo de dia vou ter hoje.

Posso reclamar porque está chovendo ou agradecer às águas por lavarem a poluição. Posso ficar triste por não ter dinheiro ou me sentir encorajado para administrar minhas finanças, evitando o desperdício. Posso reclamar sobre minha saúde ou dar graças por estar vivo.

Posso me queixar dos meus pais por não terem me dado tudo o que eu queria ou posso ser grato por ter nascido. Posso reclamar por ter que ir trabalhar ou agradecer por ter trabalho. Posso sentir tédio com o trabalho doméstico ou agradecer a Deus por ter um teto para morar.

Posso lamentar decepções com amigos ou me entusiasmar com a possibilidade de fazer novas amizades. Se as coisas não saíram como planejei posso ficar feliz por ter hoje para recomeçar.

O dia está na minha frente esperando para ser o que eu quiser. E aqui estou eu, o escultor que pode dar forma.

Tudo depende só de mim."

Charles Chaplin

RESUMO

Este trabalho pretende mostrar a importância dos testes num processo de desenvolvimento de software, fazendo uma descrição sobre a atividade de testes e os diversos tipos de testes, dando ênfase principalmente aos testes funcionais. Serão apresentadas diversas técnicas de testes funcionais e como essas técnicas são utilizadas na prática nos sistemas desenvolvidos pela Sinfo/UFRN – SIPAC, SIGAA e SIGRH. Em seguida será analisado o uso de uma ferramenta de gerenciamento de software, o IProject, mostrando seu projeto inicial, como ele pode auxiliar os gerentes na administração do fluxo das tarefas e como pode agilizar o processo de desenvolvimento. Será detalhado também o aprimoramento do projeto ao adicionar a gerência das atividades de testes à ferramenta IProject, abordando as tecnologias usadas, voltadas ao desenvolvimento de sistemas web orientado a objetos, as alterações realizadas no projeto inicial do IProject, os novos casos de uso que surgiram com o incremento da nova funcionalidade e por fim, uma análise dos resultados obtidos pelos gerentes.

Palavras-chave: Testes. Testes funcionais. Processo de desenvolvimento. Ferramenta de gerenciamento de software.

LISTA ABSTRACT

This work aims to show the importance of testing in a software development process, presenting a description of the testing activities and the various types of tests, emphasizing mainly the functional tests. It will be presented several techniques for functional tests and how these techniques are used in practice in the systems developed by Sinfo/ UFRN – SIPAC, SIGAA e SIGRH. Then, the use of a software management tool, the IProject, will be analyzed, showing its initial project, how it can assist the managers in the administration of the task flows and how to streamline the development process. It will also detail the improvement of the project by adding management of the testing activities to the IProject tool, addressing the used technologies, geared to the development of object-oriented web systems, the changes made in the IProject initial project, the new use cases that emerged with the rise of the new functionality and finally an analysis of the results achieved by managers.

Key words: Tests. Functional tests. The development process. Software management tool.

LISTA DE ILUSTRAÇÕES

Figura 1 – Testes no contexto da Engenharia de Software.....	13
Figura 2 – Incidência de erros nas fases de desenvolvimento de um projeto de software.	14
Figura 3 - Fluxo de informação de teste.....	16
Figura 4 – Técnica de Teste de Caixa Branca	18
Figura 5 – Técnica de Teste de Caixa Preta	19
Quadro 1 – Estrutura de uma Tabela de Decisão	27
Quadro 2 – Exemplo de uma Tabela de Decisão.....	27
Quadro 3 – Tabela com Casos de Testes.....	28
Figura 6 – Notação do Grafo Causa-Efeito	29
Figura 7 – Grafo de Causa-Efeito.....	30
Quadro 4 - Tabela de Decisão	30
Figura 8 – Organograma da Superintendência de Informática.....	33
Figura 9 - Relacionamento entre os sistemas da UFRN.....	35
Figura 10 – Diagrama de Casos de Uso do IProject.....	37
Figura 11 – Diagrama de Classes do IProject	40
Figura 12 - Diagrama de Atividades do IProject.....	41
Figura 13 – Arquitetura de software em camadas	49
Figura 14 – Arquitetura MVC	50
Figura 15 – Novo Diagrama de Classe do IProject	52
2.1.1 CASOS DE TESTE	24
2.1.2 CRITÉRIOS DE PARADA DE TESTES	25
2.2 TESTES FUNCIONAIS	25
2.2.1 TESTE DE EQUIVALÊNCIA DE CLASSE	26
2.2.2 TESTE DE VALOR LIMITE	26
2.2.3 TESTE POR TABELA DE DECISÃO	27
2.2.4 TESTE DE GRAFO DE CAUSA-EFEITO	28
3 PROJETO - FERRAMENTA DE GERENCIAMENTO DE SOFTWARE	32
3.1 SUPERINTENDÊNCIA DE INFORMÁTICA – UFRN	32
3.2 SUPERINTENDÊNCIA DE SISTEMAS	33
3.3 IPROJECT	34
3.4 SISTEMAS GERENCIADOS PELO IPROJECT	35
3.5 DIAGRAMA DE CASOS DE USO	36
3.6 DIAGRAMA DE CLASSES	39
3.7 DIAGRAMA DE ATIVIDADES	40
4 TESTES FUNCIONAIS NOS SISTEMAS DA UFRN	43
4.1 TESTES DE USABILIDADE	43
4.2 TESTES DE CONSISTÊNCIA EM FORMULÁRIOS	43
4.3 TESTES DE TEMPO DE RESPOSTA E INTEROPERABILIDADE	43
4.4 PERSONALIZAÇÃO DE LAYOUT	45
4.5 TESTES DE NAVEGAÇÃO	45
4.6 TESTES DE INTEGRIDADE DOS DADOS	46
4.7 TESTES DOS CASOS DE USO	46
5 PROJETO DE INCORPORAÇÃO DOS TESTES FUNCIONAIS AO IPROJECT	47
5.1 TECNOLOGIAS UTILIZADAS	47
5.2 A ARQUITETURA DE SOFTWARE EM CAMADAS	47
5.3 A ARQUITETURA MODEL-VIEW-CONTROLLER (MVC)	49
5.4 A JARNA SERVER FACES (JSF)	50

5.1.4 ENTERPRISE JAVA BEANS (EJB).....	50
5.1.5 HIBERNATE	51
5.1.6 POSTGRESQL	51
5.2 NOVO DIAGRAMA DE CLASSES DO IPOJECT	51
5.3 CASO DE USO ‘SOLICITAR TESTES’	52
5.4 CASO DE USO ‘TESTE REALIZADO’	53
6 CONCLUSÕES.....	56
REFERÊNCIAS	57

necessidade de utilizar um software para auxiliar na gestão de projetos dos sistemas. Assim, para suprir essa necessidade, foi utilizado o software livre Bugzilla, que gerenciava o cadastro de erros e tarefas.

Em seguida começou o desenvolvimento dos sistemas de área avaliativa. Com esse projeto apareceram algumas dificuldades como: a grande quantidade de sistemas e serviços desenvolvidos, os prazos cada vez mais curtos, uma equipe reduzida e o aparecimento de novas funcionalidades. Deixando aplicar o processo de desenvolvimento e melhorar o gerenciamento das tarefas executadas uma vez que o Bugzilla não mais atendia as necessidades, a Diretoria do Sistema resolveu testar outros sistemas de gerenciamento como o Jira e Jira. Mas muitas das necessidades da equipe não foram supridas e então foi criado o IPOject, um novo projeto para gerenciamento de projetos em geral, não sendo direcionado especificamente para projetos de software. Esse fato motivou a equipe a criar seu próprio sistema para gerenciamento, de modo que atendessem aos anseios de todos os envolvidos no projeto, surgindo assim a versão inicial do IPOject.

Com o IPOject a diretoria do sistema conseguiu um maior controle sobre as tarefas e as responsabilidades, melhorando significativamente o gerenciamento e a produtividade do fluxo de trabalho.

Uma dificuldade encontrada foi a ocorrência de problemas nas operações dos sistemas em ambiente de produção e a falta de controle de acesso e implantação de novos sistemas, o que causava perda de tempo e afetava as equipes de desenvolvimento. Isso era decorrente dos testes serem realizados apenas pelos desenvolvedores, não havendo uma equipe responsável por tal tarefa.

Para resolver esse problema e melhorar a qualidade de software, os gerentes começaram a verificar e validar a lógica de scripts realizados nos sistemas antes mais cuidadosos e mais precisos, com a criação de uma equipe qualificada. Estudos foram realizados e alguns fatores levaram a diretoria a adotar a criação de uma equipe de testes, tais como:

- Exigências cada vez maiores dos usuários dos sistemas;
- Custos envolvidos associados às falhas;
- Gerentes não terem um bom controle sobre os erros.

1 INTRODUÇÃO

Em 2003 a UFRN e sua Superintendência de Informática iniciaram um projeto para informatização da gestão de suas áreas acadêmica, administrativa e de recursos humanos, iniciando pelo desenvolvimento dos sistemas da área administrativa. Logo surgiu a necessidade de utilizar um software para auxiliar na gerência de projetos dos sistemas. Assim, para suprir essa necessidade, foi utilizado o software livre Bugzilla, que gerenciava o cadastro de erros e tarefas.

Em seguida começou o desenvolvimento dos sistemas da área acadêmica. Com esse início apareceram algumas dificuldades como: a grande quantidade de sistemas a serem desenvolvidos, os prazos cada vez mais curtos, uma equipe reduzida e o aparecimento de novas funcionalidades. Desejando agilizar o processo de desenvolvimento e melhorar o acompanhamento das tarefas executadas, uma vez que o Bugzilla não mais atendia as expectativas, a Diretoria de Sistemas resolveu testar outros sistemas de gerenciamento como o DotProject. Mas muitas das necessidades da equipe não foram supridas: o DotProject era direcionado mais para gerenciamento de projetos em geral, não sendo direcionado especificamente para projetos de software. Este fato motivou a equipe a criar seu próprio software para gerenciamento, de modo que atendessem aos anseios de todos os envolvidos nos projetos, surgindo assim a versão inicial do IProject.

Com o IProject a diretoria de sistemas conseguiu um maior controle sobre as tarefas e seus respectivos responsáveis, melhorando significativamente o gerenciamento e a dinamicidade do fluxo de trabalho.

Outra dificuldade encontrada foi a ocorrência de problemas nas operações dos sistemas durante a fase de implantação, o que causava perda de tempo e sufocava as equipes de suporte e desenvolvimento. Isso era decorrente dos testes serem realizados apenas pelos próprios desenvolvedores, não havendo uma equipe responsável por tal tarefa.

Para resolver esse problema e melhorar a qualidade do software, os gerentes começaram a amadurecer a idéia de serem realizados nos sistemas testes mais cuidadosos e bem planejados através de uma equipe qualificada. Estudos foram realizados e alguns fatores ajudaram a diretoria a aderir à criação de uma equipe de testes, tais como:

- Exigências cada vez maiores dos usuários dos sistemas;
- Custos envolvidos associados às falhas;
- Gerentes não terem um bom controle sobre os erros;

- Previsão de que com o desenvolvimento de outros módulos os erros poderiam aumentar consideravelmente;

- Além dos testes serem uma tendência do mercado.

Com a inserção da nova fase ao fluxo do projeto, tornou-se necessário adicionar seu controle ao IProject. Com este objetivo, o projeto foi aprimorado: novos diagramas foram desenvolvidos e além de atender a nova demanda, problemas detectados na versão anterior foram resolvidos.

Para uma melhor organização, este trabalho está dividido em cinco capítulos: no capítulo 2 são apresentados os conceitos básicos de testes de software, os principais tipos de testes e análises com algumas técnicas de testes funcionais. No capítulo 3 é realizada uma análise da ferramenta de gerenciamento de software IProject, os órgãos responsáveis pelo seu desenvolvimento, os sistemas gerenciados por essa ferramenta e uma descrição de sua implementação. No capítulo 4 descrevem-se os testes funcionais nos sistemas desenvolvidos na UFRN, mostrando os procedimentos realizados para se obter êxito no controle da qualidade dos testes. O capítulo 5 retrata o projeto para adicionar as novas funcionalidades ao IProject com o intuito de contemplar a atividade de testes. E no capítulo 6 contém relatos sobre as conclusões finais e os resultados obtidos pela UFRN com a inserção dos testes funcionais ao IProject.

2.1 TESTES DE SOFTWARE

É um processo dentro da Engenharia de Software para encontrar defeitos. Ele ocorre por vários processos da Engenharia de Software, envolvendo ações que vão do levantamento de requisitos até a execução do teste propriamente dito. Testar um software significa verificar se ele é uma criação controlada se o seu comportamento corre de acordo com o especificado.

Na Engenharia de Software, Teste é uma parte importante dentro da Garantia da Qualidade de Software (GQS). A área de GQS é composta pelos elementos Ocorrência de Defeito (OD) – que gerencia, controla e registra os artefatos que compõe o software; Gerência de Requisitos – que gerencia, controla e registra as necessidades do sistema (requisitos) e dos

2 TESTES FUNCIONAIS DE SOFTWARE

Testes em software sempre ficaram em segundo plano num processo de desenvolvimento: eram vistos como um trabalho tedioso e que não se deveria investir muito tempo nem dinheiro. Com a exigência cada vez maior do mercado, o aumento da disputa entre empresas de software e os prejuízos que um erro pode causar estão fazendo com que as empresas dêem grande valorização à atividade de testes. Segundo Pressman (1995, p.786):

“O destaque crescente do software como elemento de sistema e os “custos” envolvidos associados às falhas de software são forças propulsoras para uma atividade de testes cuidadosa e bem planejada. Não é incomum que uma organização de software gaste 40% do esforço de projeto total em teste.”

Hoje em dia teste de software é uma área importante, e tem se destacado principalmente pela importância que as empresas estão dando a produtos com alto grau de qualidade. Em muitos casos, testes podem salvar uma empresa, seja pela quantidade de erros encontrados mais cedo ou por problemas evitados antes da aplicação entrar em produção.

A área de Testes se tornou uma ciência: atualmente existem congressos, simpósios e vários tipos de certificações. É uma área que vem crescendo rapidamente, e isso pode ser visto pelos inúmeros empregos que estão sendo oferecidos no mercado.

2.1 TESTE DE SOFTWARE

É um processo dentro da Engenharia de Software para encontrar defeitos. Ele permeia por vários processos da Engenharia de Software, envolvendo ações que vão do levantamento de requisitos até a execução do teste propriamente dito. Testar um software significa verificar através de uma execução controlada se o seu comportamento corre de acordo com o especificado.

Na Engenharia de Software, Teste é uma parte importante dentro da Garantia da Qualidade de Software (GQS). A área da GQS é composta pelos elementos Gerência de Configuração – que gerencia, controla e registra os artefatos que compõe o software; Gerência de Requisitos – que gerencia, controla e registra as necessidades do sistema (requisitos) e dos

usuários; e Testes – que gerencia, planeja, controla e executa os testes, garantindo a implementação. Testes apenas controlam a qualidade, mas não a garantem. A garantia é realizada pelas outras áreas da GQS. Na figura 1 pode-se ver onde se situam os Testes de Software no contexto da Engenharia de Software.

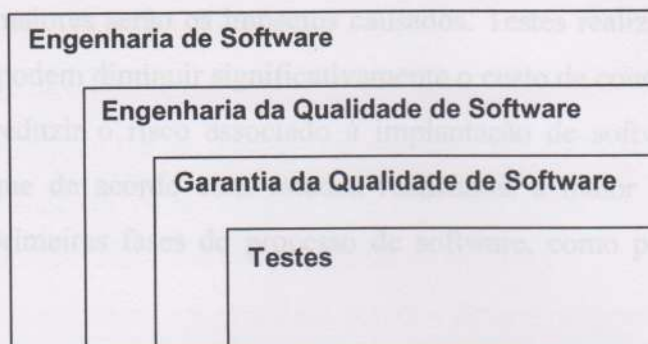


Figura 1 – Testes no contexto da Engenharia de Software
Fonte: Molinari, 2008

2.1.1 OBJETIVOS

O objetivo de um teste é descobrir se um programa está em desacordo com os requisitos funcionais e não funcionais. A maioria das pessoas pensa que o teste de software serve para demonstrar o correto funcionamento de um programa, quando na verdade ele é utilizado como um processo da engenharia de software para encontrar defeitos. Os testes mostram a presença, e não a ausência de defeitos.

Em seu livro sobre teste de software, Myers apud Pressman (1995, p.788) afirma que:

1. A atividade de teste é o processo de executar um programa com a intenção de descobrir um erro.
2. Um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto.
3. Um teste bem sucedido é aquele que revela um erro ainda não descoberto.

Com o resultado dos testes consegue-se outro objetivo que é proporcionar uma boa indicação da confiabilidade do software e alguma indicação da qualidade do software como um todo.

2.1.2 QUANDO TESTAR E QUEM REALIZA OS TESTES?

O ideal é utilizar metodologias que se preocupem com testes em todas as fases do desenvolvimento. Os erros podem estar já nas primeiras fases do projeto e quanto mais tarde forem descobertos, maiores serão os impactos causados. Testes realizados no início do ciclo de vida do software podem diminuir significativamente o custo de conclusão e manutenção do software, além de reduzir o risco associado à implantação de software de má qualidade. Principalmente porque de acordo com estudos realizados, a maior incidência de erros se encontra logo nas primeiras fases do processo de software, como pode ser visualizado na figura 2.

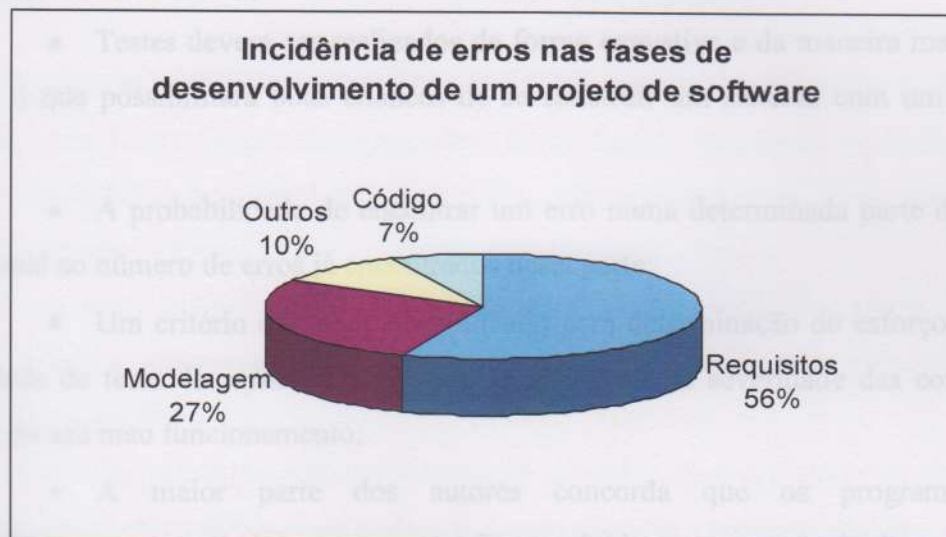


Figura 2 – Incidência de erros nas fases de desenvolvimento de um projeto de software.
Fonte: Tozelli, 2008b

Testes devem ser realizados por uma variedade de pessoas. Os desenvolvedores são responsáveis por testar unidades individuais antes de realizarem a integração destas unidades ao projeto. Eles devem ter interesse em demonstrar que o código está livre de falhas. O Grupo da Garantia de Qualidade (ou Equipe de Testes) fica encarregado de executar os testes de caixa preta, onde o objetivo principal é descobrir o máximo de falhas possível e encaminhar para que os desenvolvedores corrijam os erros descobertos.

2.1.3 PRINCÍPIOS DOS TESTES DE SOFTWARE

Abaixo estão relacionados alguns dos itens que são comuns a todos os autores e pesquisadores do assunto “Teste de Software” e que descrevem os fundamentos e princípios desta atividade.

- Todos os testes devem ser seguidos de acordo com os requisitos do projeto.
- Conforme a Lei de Pareto, lei proposta em 1897 pelo economista italiano Vilfredo Pareto (1848-1923), onde, segundo ele, 80% dos erros podem ser localizados em 20% do projeto, geralmente nos módulos principais do sistema;
- Testes começam em pequenas partes e devem progredir de forma a se atingir as grandes partes do sistema;
- Testes devem ser realizados de forma exaustiva e da maneira mais completa possível, o que possibilitará boas chances de se construir um sistema com um mínimo de falhas;
- A probabilidade de encontrar um erro numa determinada parte do sistema é proporcional ao número de erros já encontrados nesta parte;
- Um critério que pode ser utilizado para determinação do esforço a ser gasto na atividade de teste de *software* é verificar qual o grau de severidade das conseqüências advindas do seu mau funcionamento;
- A maior parte dos autores concorda que os programas devem, preferencialmente, ser testados por pessoas não envolvidas no processo de desenvolvimento, por uma equipe independente. Pode haver também a interação dos desenvolvedores com a equipe independente, justificando as decisões tomadas durante o projeto. Esta abordagem ajuda na revisão do projeto.

2.1.4 FLUXO DE TESTES

O fluxo de informação referente à atividade de teste pode ser descrito de acordo com a figura 3.

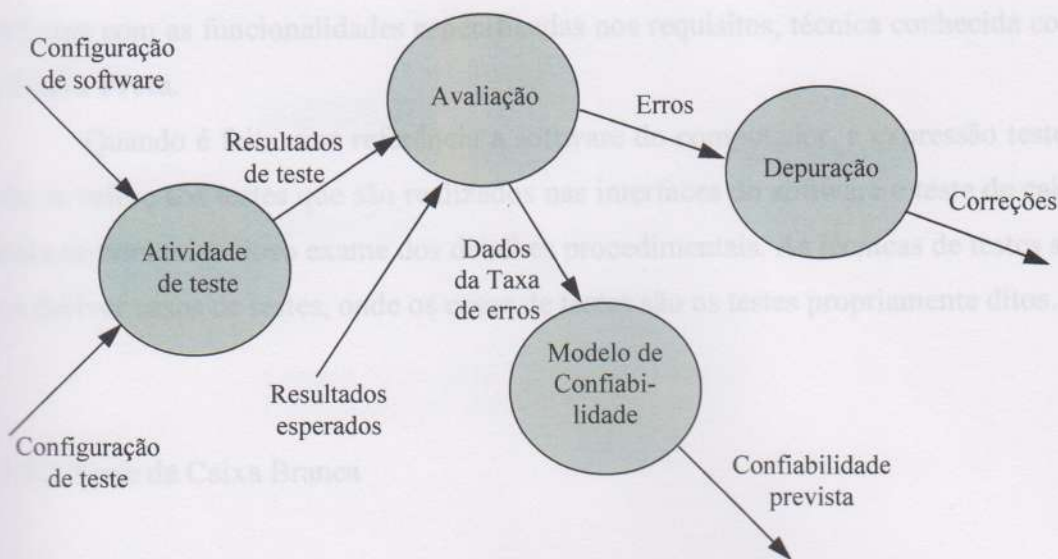


Figura 3 - Fluxo de informação de teste

Fonte: Pressman, 1995

O fluxo se inicia com os seguintes dados de entrada: a configuração do software que é composta pela Especificação dos Requisitos, Especificação do projeto e o código-fonte; e a configuração de testes que inclui o Plano e o Procedimento de Testes, Ferramentas de Testes que serão usadas e os Casos de Testes e seus resultados esperados.

Com esses dados os testes são realizados e todos os resultados são avaliados. Essa avaliação é feita comparando os resultados dos testes com os testes esperados. Quando forem encontrados dados errados, estes irão para depuração e correção.

À medida que os resultados dos testes vão sendo avaliados pode-se ter uma indicação da qualidade e da confiabilidade do software. Como também a regularidade com que ocorrem os erros graves.

2.1.5 TÉCNICAS DE TESTES

Qualquer produto trabalhado por engenharia pode ser testado de duas maneiras: primeiro verificando se as operações internas estão sendo executadas de acordo com as especificações, e se todos os componentes internos foram adequadamente testados, esta técnica é conhecida como **Teste de Caixa Branca**; e segundo testando o projeto como um todo, procurando saber como o sistema irá trabalhar e verificando se os testes estão em

conforme com as funcionalidades especificadas nos requisitos, técnica conhecida como **Teste de Caixa Preta**.

Quando é feita uma referência a software do computador, a expressão teste de caixa preta se refere aos testes que são realizados nas interfaces do software e teste de caixa branca baseia-se num minucioso exame dos detalhes procedimentais. As técnicas de testes são usadas para derivar casos de testes, onde os casos de testes são os testes propriamente ditos.

Figura 4 – Técnica de Teste de Caixa Branca
Fonte: Dias Neto, 2002

2.1.5.1 Teste de Caixa Branca

2.1.5.1.1 Teste de Caixa Branca

Esta técnica testa o comportamento interno do projeto. O testador trabalha diretamente com o código-fonte e verifica se as operações internas são executadas como especificado, e se todos os componentes internos foram adequadamente testados. O teste caixa branca é desenvolvido analisando-se o código fonte e elaborando-se os casos de teste para que cubram todas as possibilidades do componente de software.

Usando o teste de caixa branca podem ser extraídos casos de testes que:

- Garantam que todos os caminhos independentes dentro de um módulo tenham sido exercitados pelo menos uma vez;
- Exercitem todas as decisões lógicas para valores falsos ou verdadeiros;
- Executem todos os laços em suas fronteiras e dentro de seus limites operacionais;
- E exercitem as estruturas de dados internas para garantir a sua validade.

Exemplos de técnicas de caixa branca são o teste do caminho básico e teste de estrutura de controle.

A técnica de teste de Caixa-Branca é recomendada para as fases iniciais do processo de testes. Um exemplo de teste caixa branca é o teste de Unidade. A responsabilidade principal desse teste fica a cargo dos desenvolvedores do software, que por sua vez conhecem bem o código-fonte produzido.

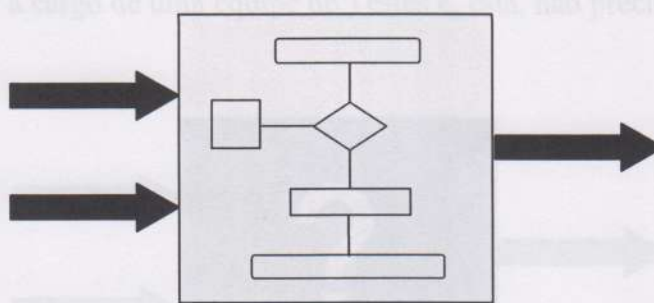


Figura 4 – Técnica de Teste de Caixa Branca
Fonte: Dias Neto, 2007

Figura 5 – Técnica de Teste de Caixa Preta
Fonte: Dias Neto, 2007

2.1.5.2 Teste de Caixa Preta

Testa o projeto como um todo, analisando como o sistema irá trabalhar e se está em conformidade com as funcionalidades especificadas nos requisitos. O teste de caixa preta concentra-se nos requisitos funcionais do software. Ele não é outra opção em relação ao teste de caixa branca, é, sim, uma abordagem que complementa o teste de caixa branca descobrindo outra classe de erros diferentes dos encontrados com este último.

O teste de caixa preta procura descobrir erros que ocorrem nas seguintes categorias: funções incorretas ou ausentes, erros de interface, erros no acesso a banco de dados externos, erros de desempenho e de inicialização e término.

Para extrair os casos de testes usando o teste de caixa preta os seguintes critérios precisam ser satisfeitos:

- Os casos de testes que reduzem, numa contagem que seja maior que 1, o número de casos de testes adicionais que devem ser projetados para se conseguir testes razoáveis;
- E casos de testes que nos dizem algo sobre a presença ou ausência de classes de erros, em vez de um erro associado somente ao teste específico que se tem em mãos.

Os testes de equivalência de classe e valor limite são exemplos de técnicas de caixa preta. No tópico sobre Testes Funcionais, serão abordadas algumas dessas técnicas e outros detalhes sobre os testes de caixa preta.

O teste de caixa preta tende a ser aplicado durante as fases finais do processo de testes. Um exemplo de teste de caixa preta é o teste Funcional. A responsabilidade principal

desse teste pode ficar a cargo de uma equipe de Testes e, esta, não precisa conhecer o código-fonte implementado.



Figura 5 – Técnica de Teste de Caixa Preta
Fonte: Dias Neto, 2007

2.1.6 ESTRATÉGIAS DE TESTES

Ao pensar em realizar testes de software, dificilmente se utiliza apenas um tipo de teste. Para testar uma aplicação são realizados vários tipos de testes nas diversas fases do processo de desenvolvimento do software. Assim a aplicação pode ser vista nos mais variados ângulos tornando mais difícil escapar alguma falha.

Por isso, sempre é bom escolher uma estratégia de testes que forneça desde testes de baixo nível (validando o código fonte através do teste unitário) até testes de alto nível (validando funcionalidades do sistema contra requisitos do cliente usando testes funcionais).

Uma **Estratégia de Testes** é um conjunto de técnicas de testes com etapas bem planejadas que resultarão em uma construção bem sucedida do projeto de software. Uma estratégia de teste de software proporciona aos usuários finais, desenvolvedores e grupo de garantia de qualidade um roteiro para ajudar a conduzir o teste. Esse roteiro descreve os passos a serem dados como parte da atividade de teste, quando esses passos são planejados, e depois executados, e quanto esforço, tempo e recursos são exigidos.

O objetivo da estratégia de teste de software é assegurar que o projeto construído execute uma função específica de acordo com um conjunto de atividades (verificação), e que atinja as exigências do cliente seguindo um conjunto diferente de atividades (validação).

Segundo Boehm, 1981 apud Pressman (1995, p.836) essa questão tem outro sentido:

- Verificação: “Estamos construindo certo o produto?”
- Validação: “Estamos construindo o produto certo?”

2.1.7 TIPOS DE TESTES

Como apresentado, uma boa estratégia de testes possui vários tipos de testes para as diversas fases do projeto. A seguir serão citados alguns dos principais tipos de testes (o tipo de teste funcional será detalhado no próximo tópico).

2.1.7.1 Teste de Unidade ou Teste Unitário

Concentra-se esforço de verificação da menor unidade de projeto de software. O objetivo é o de encontrar falhas de funcionamento dentro de uma pequena parte do sistema funcionando independentemente do todo, podendo, dessa forma, ser realizado em paralelo para múltiplas unidades. No contexto da engenharia de software orientada a objetos, uma unidade é uma classe e um teste unitário é o teste dessa classe isoladamente.

O teste unitário em geral é considerado um auxílio da etapa de codificação, onde depois que o código foi desenvolvido, revisado e verificado a fim de conseguir uma sintaxe correta, inicia-se o projeto do caso de teste de unidade. Cada caso de teste deve estar associado a um conjunto de resultados esperados.

É realizado pelo desenvolvedor durante o desenvolvimento da unidade. Lida diretamente com o código-fonte, sendo assim se baseia em testes de caixa branca.

2.1.7.2 Teste de Integração

Após os testes de unidade, todas as classes devem ser testadas em integração. Neste estágio é verificado se cada componente interage de maneira satisfatória quando agregados e se todas as interfaces estão corretas.

O objetivo é, a partir dos módulos testados no nível de unidade, construir a estrutura de programa que foi determinada pelo projeto.

2.1.7.3 Teste de Regressão

Algumas vezes quando algum erro é corrigido, é necessário refazer o teste dos componentes. O Teste de Regressão é uma re-execução de alguma parte do componente para assegurar que mudanças realizadas na correção dos erros não produziram erros não-intencionais. Um novo teste ocorre para as funções do sistema que foram afetadas, componentes que foram modificados e os que podem ser afetados por uma mudança.

Para efeito de aumento de produtividade e de viabilidade dos testes, é recomendada a utilização de ferramentas de automação de testes, de forma que, sobre a nova versão ou ciclo de teste, todos os testes anteriores possam ser re-executados com maior agilidade.

2.1.7.4 Teste de Validação

Os testes de validação se iniciam após o término dos testes de Integração. Uma validação é bem sucedida quando um software funciona de maneira razoavelmente esperada pelo cliente. A validação de um software consiste numa série de casos de teste caixa-preta que demonstram conformidade com os requisitos. O software é completamente integrado como um sistema e erros de interface entre componentes de software foram descobertos e corrigidos. O foco é nas ações visíveis ao usuário e nas saídas reconhecíveis pelo usuário.

2.1.7.5 Testes Alfa e Beta

Uma série de testes de aceitação que permitem ao usuário final validar todos os requisitos. Um teste de aceitação pode ir desde testes informais a testes executados de forma planejada e sistemática. Este teste permite aos usuários finais descobrir erros e defeitos que só eles podem encontrar. Na sua grande maioria os construtores de softwares comerciais usam um processo denominado de teste alfa e teste Beta. Um **Teste Alfa** é conduzido em um ambiente controlado, geralmente no ambiente de desenvolvimento. Usuários finais são solicitados a usar o sistema como se estivessem trabalhando naturalmente, enquanto os

desenvolvedores estão anotando erros e problemas de utilização. **Testes Beta**, ao contrário, são conduzidos em um ou mais ambientes do usuário e os desenvolvedores não estão presentes. Neste caso, são os próprios usuários finais que anotam todos os erros e problemas de utilização. Eles encaminham estes erros para os desenvolvedores, para que façam os consertos.

2.1.7.6 Teste de Sistema

É uma série de diferentes testes com o propósito de pôr a prova o sistema baseado em computador. Mesmo cada teste tendo uma finalidade diferente, todo o trabalho deve verificar se todos os elementos do sistema foram adequadamente integrados e realizam as funções atribuídas. Normalmente é realizado quando o software está funcionando totalmente.

Em seguida serão apresentados alguns testes de Sistemas:

- **Teste de Recuperação:** testa o mecanismo de recuperação do software quando ele falha. É um teste que força o software a falhar de diversas maneiras e verifica se a recuperação é adequadamente executada. Se a recuperação for automática (realizada pelo próprio sistema) algumas avaliações são realizadas como a reinicialização, recuperação de dados, mecanismos de *checkpointing*, etc. E se a recuperação exigir a participação humana é avaliado se o tempo médio até o reparo se encontra dentro dos limites aceitáveis.
- **Teste de Segurança:** tenta verificar se todos os mecanismos de proteção embutidos num sistema o protegerão, de fato, de acessos indevidos. No teste, o analista de sistemas faz o papel de um hacker e tenta de várias maneiras acessar indevidamente no sistema.
- **Teste de Estresse:** é realizado para confrontar os programas com situações anormais de quantidade, frequência e volume de dados, pedidos e respostas. Conduz uma avaliação de um sistema ou um componente além ou aquém dos limites que suporta ou não para determinar quando falha ou não, quando um pico é atingido. É semelhante ao Teste de Desempenho, porém os testes são submetidos a uma carga bem maior de dados.
- **Teste de Desempenho ou Performance:** testa se o tempo de resposta e a utilização de recursos estão conforme os requisitos de desempenho. Ocorre ao longo de todos os passos do processo de teste, inclusive no teste de unidade. Porém só quando todos os elementos de sistema estão plenamente integrados é que o desempenho real de um sistema pode ser

verificado. Este tipo de teste identifica onde residem os gargalos dos sistemas, se é na própria aplicação ou na infra-estrutura em que ela está implementada.

2.1.8 PLANO DE TESTES

Principal artefato da atividade de testes. Descreve detalhadamente o processo de testes de determinada aplicação ou nova característica adicionada a uma versão antiga. É composto pelos requisitos de testes, casos de testes, cenários de testes a serem utilizados e quais quer outros documentos que sejam pertinentes ao planejamento. Por uma questão de integridade das informações de teste, um plano de teste deve-se manter sempre atualizado.

Dependendo de onde é utilizado, o plano de teste possui várias funções, entre elas tem-se:

- Suportar o desenvolvimento da qualidade dos produtos, de forma sábia, sincronizada com as decisões;
- Descrever e justificar a estratégia de teste para com os requisitos do produto a ser testado;
- Suportar o início e a organização do projeto de teste, incluindo aí preparações, pessoal, planejamento de tarefas etc.;
- Suportar o gerenciamento diário e a evolução do projeto de teste e da estratégia;
- Suportar a efetiva coordenação, colaboração e outros relacionamentos entre os membros da equipe de teste e o resto do projeto;
- Identificar e gerenciar quaisquer riscos ou questões que possam impactar no projeto.

Um plano de teste pode seguir algumas etapas durante a sua elaboração e utilização:

- Definição do escopo: o que se deve testar em nível macro;
- Identificação de requisitos e casos de teste: identificando os requisitos de forma hierárquica e descritiva e identificando os casos de teste que abranjam os requerimentos de teste em detalhes, incluindo aí os passos de execução de testes;
- Identificação das prioridades: o que se deve testar e em qual ordem;

- Definição da estratégia: quais técnicas de teste serão utilizadas e em quais requisitos;
- Identificação de recursos: quem fará o que e o que será utilizado (hardware, software etc.);
- Elaboração de um cronograma de testes;
- Gerar documento de plano de testes com a documentação formal e revisada, com as devidas análises do planejamento de teste;
- Atualização constante do plano de teste com os resultados finais de cada teste e com os defeitos encontrados.

Existem diversos padrões específicos de elaboração e uso do plano de testes. Mas por mais vantagens que se tem ao usar um modelo formal, na prática as empresas brasileiras não usam um padrão de fato. Elas criam seu próprio padrão fazendo uma mescla dos vários padrões que se adapte à sua necessidade.

2.1.9 CASOS DE TESTE

É o teste a ser executado. Nos casos de testes estão inseridos os procedimentos necessários para a execução de um determinado teste no sistema.

Da mesma forma que o plano de teste, o caso de teste também pode seguir padrões, mas em geral, na prática, as empresas criam seus próprios casos de testes de maneira que atenda às suas necessidades. Um caso de teste pode ser composto por:

- Descrição do teste e em que etapa do processamento o teste deve ser realizado;
- Passos para execução dos testes;
- Resultado esperado para cada passo ou geral;
- Dados com as informações utilizadas no teste;
- Pré-condições, se existirem para execução dos testes;
- Pós-condições, se existirem para execução dos testes;
- Resultado da execução, indicando se o teste passou ou não e por quê.

Podendo ser usado para cada passo;

- Informações e configurações sobre o ambiente para realização dos testes;

- Informações sobre prazo, como cronograma, data etc.;
- Informações sobre quem realizou o teste, datas etc.;
- Status do teste informando se o teste foi criado, atualizado, finalizado, aprovado, executado, etc.

2.1.10 CRITÉRIOS DE PARADA DE TESTES

Não tem como garantir que todos os testes foram realizados. Mas pode-se levar em consideração alguns fatores que ajudam a decidir, uma vez que:

- Fundamentalmente é decisão gerencial porque diz respeito a recursos, alocação de pessoal, etc.;
- Obrigatoriamente é decisão comercial, porque influencia o custo, prazo, etc.;
- Necessariamente é decisão do cliente, quando identifica o nível de qualidade necessária para o produto.

2.2 TESTES FUNCIONAIS

É um tipo de teste de software que utiliza técnicas de teste de caixa preta e que mostra se a aplicação funciona ou não em tudo aquilo que ela se propõe a atender em termos de funcionalidades, ou seja, testa as funcionalidades presentes na documentação, verificando se funcionam ou não como especificado. Tem como objetivo medir a qualidade funcional de componentes de um sistema.

O teste funcional tem grande importância não só para testar a funcionalidade suportada pelo sistema, mas para testar a adequação e o nível da qualidade associado à aplicação em si. Atualmente ignorar testes funcionais é ignorar o negócio de sua empresa e conseqüentemente ignorar sua sobrevivência no mundo corporativo.

A seguir são apresentadas algumas das técnicas empregadas nesse tipo de teste.

2.2.1 TESTE DE EQUIVALÊNCIA DE CLASSE

É uma técnica simples e pode ser usada de forma intuitiva pelos testadores. Tem o objetivo de reduzir o número de casos de testes a um montante gerenciável, mas mantendo um elevado grau de cobertura. Para entender melhor o funcionamento dessa técnica uma análise pode ser realizada no sentido da equivalência na teoria dos conjuntos. Equivalência é a existência de uma relação entre todos os elementos de uma classe. Nesse caso, os testes serão executados selecionando-se uma variável de cada classe de equivalência em um determinado caso de teste. Com isso, o número de casos de teste será equivalente ao número de partições encontradas. A idéia de equivalência é garantir que se está utilizando um elemento representativo de cada partição e que todos os diferentes tipos de elementos da classe serão testados. Assim, evita-se as redundâncias.

Veja o seguinte exemplo: considere as regras de negócio de uma empresa, que definem a idade de contratação de um funcionário:

- Faixa etária de 0-15: não contrata funcionário;
- Faixa etária de 16-17: contrata funcionário em tempo parcial;
- Faixa etária de 18-54: contrata funcionário em tempo integral;
- Faixa etária de 55-99: não contrata funcionário.

Para testar se o funcionário pode ser contratado ou não pelo sistema devem ser testados os valores de 0 a 99, mas não faz tanto sentido testar todos esses valores. Analisando a classe com a faixa etária de 0 a 16, não é necessário testar para 0, 1, 2, 3, 4, 5, 6, até 16, pois vão ter os mesmos resultados. Então, usando a técnica de equivalência para essa classe há uma redução na quantidade de testes e os testes podem ser feitos para apenas um número, por exemplo, 10. Fazendo isso para as outras classes, diminui-se a quantidade de testes e são usados apenas 4 valores (ex.: 10, 17, 40, 65).

2.2.2 TESTE DE VALOR LIMITE

Também é uma técnica simples e que objetiva reduzir o número de casos de teste. Essa técnica tem a característica básica de testar os valores limites ou de fronteiras das classes, analisando como se comportaria a aplicação para esses valores limites. É um

complemento a técnica de Equivalência de Classe, onde em vez de selecionar qualquer valor de uma classe de equivalência, são selecionados valores nas extremidades da classe.

Tomando o mesmo exemplo da contratação de funcionário pela faixa etária, visto na técnica de Equivalência de Classe, os seguintes valores de fronteiras poderiam ser usados para testar cada classe: $\{-1, 0, 1\}$, $\{15, 16, 17\}$, $\{17, 18, 19\}$, $\{54, 55, 56\}$ e $\{98, 99, 100\}$. Assim pode-se ver como se comportaria a aplicação para cada valor de fronteira testado.

Casos de Testes	Casado?	Bom estudante?	Resultado (Desconto)
Caso 1 (Caso de Teste 1)	Sim	Sim	60
Caso 2 (Caso de Teste 2)	Sim	Não	25
Caso 3 (Caso de Teste 3)	Não	Sim	50
Caso 4 (Caso de Teste 4)	Não	Não	0

Quadro 3 – Tabela com Casos de Testes
Fonte: Dados do Proprietário, 2008

É uma técnica para criar casos de uso e deve ser aplicada principalmente quando se tem muitas regras de negócios e, essas regras são muito complexas.

O quadro 1 mostra um exemplo de estrutura de uma Tabela de Decisão, onde se tem a representação das regras de negócio baseadas em um conjunto de condições e de acordo com as combinações dessas condições as ações serão realizadas.

	Regra 1	Regra 2	...	Regra p
Condições				
Condição 1				
Condição 2				
...				
Condição m				
Ações				
Ação 1				
Ação 2				
...				
Ação n				

Quadro 1 – Estrutura de uma Tabela de Decisão
Fonte: Molinari, 2008

Agora, um pequeno exemplo de uma tabela de decisão onde se tem uma ação e duas condições.

	Regra 1	Regra 2	Regra 3	Regra 4
Condições				
Casado?	Sim	Sim	Não	Não
Bom estudante?	Sim	Não	Sim	Não
Ações				
Desconto (\$)	60	25	50	0

Quadro 2 – Exemplo de uma Tabela de Decisão
Fonte: Molinari, 2008

Veja que as condições são perguntas e para cada regra existe um conjunto de respostas a essas perguntas. E, de acordo com esse conjunto de resposta de cada regra forma-se a realização de uma ação.

Depois que se conclui a tabela de decisão, transforma-se essa tabela em casos de testes. Que é uma tarefa bem simples, pois os casos de testes são os títulos das colunas da tabela. Pegando o exemplo do quadro 2, os seguintes casos de testes são extraídos:

Casos de Testes	Casado?	Bom estudante?	Resultado (Desconto)
Regra 1 (Caso de Teste 1)	Sim	Sim	60
Regra 2 (Caso de Teste 2)	Sim	Não	25
Regra 3 (Caso de Teste 3)	Não	Sim	50
Regra 4 (Caso de Teste 4)	Não	Não	0

Quadro 3 – Tabela com Casos de Testes
Fonte: Dados do Pesquisador, 2008

2.2.4 TESTE DE GRAFO DE CAUSA-EFEITO

Técnica que oferece uma representação entre as causas e os efeitos correspondentes. As causas são as condições de entrada e os efeitos são as ações realizadas em resposta às diferentes condições de entrada. São necessários os seguintes passos para realizar essa técnica:

- I. As causas e os efeitos são relacionados para um módulo e atribuído um identificador para cada um.
- II. Um grafo de causa-efeito é desenvolvido.
- III. Depois esse grafo é transformado em uma tabela de decisão.
- IV. E por fim, as regras da tabela de decisão são convertidas em casos de testes.

A figura 6 retrata algumas notações básicas para o grafo de causa-efeito.

• Causas

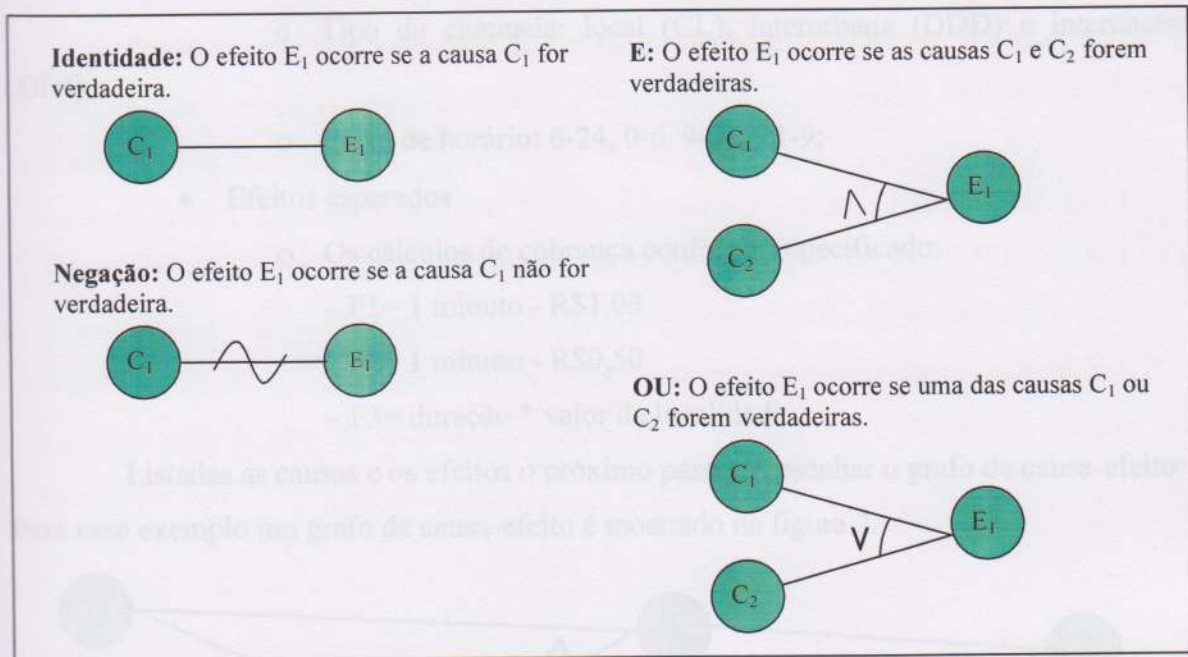


Figura 6 – Notação do Grafo Causa-Efeito
Fonte: Ferreira, 2005

Para entender melhor o uso do grafo de causa-efeito considere o exemplo de um programa de cobrança de uma empresa de telefonia com as seguintes regras:

- Os valores para cada chamada são contabilizados de acordo com o local de destino da chamada e a faixa de horário.
- Se o local de destino for o mesmo da origem (chamada local) e a faixa de horário for das 6:00 às 23:59, então valor do minuto é R\$ 1,00.
- Se for uma chamada local e a faixa de horário for 0:00 às 5:59, o valor do minuto de cada chamada é R\$0,50.
- Se o local for outro estado no país (chamada interurbana) e a faixa de horário for das 9:00 às 21:00 então o valor do minuto é calculado de acordo com o valor básico por estado.
- Se chamada interurbana e faixa de horário for das 21:00 às 9:00 o valor do minuto é fixo, sendo R\$1,00.
- Se chamada internacional o valor não depende de faixa de horário e é calculado de acordo com o valor básico por país.

A partir desses dados são relacionadas as causas e os efeitos e atribui-se um identificador:

- Causas

○ Tipo da chamada: local (CL), interurbana (DDD) e internacional (DDI);

○ Faixa de horário: 6-24, 0-6, 9-21, 21-9;

- Efeitos esperados

- Os cálculos de cobrança conforme especificado:

- F1= 1 minuto - R\$1,00

- F2= 1 minuto - R\$0,50

- F3= duração * valor da localidade.

Listadas as causas e os efeitos o próximo passo é desenhar o grafo de causa-efeito.

Para esse exemplo um grafo de causa-efeito é mostrado na figura 7.

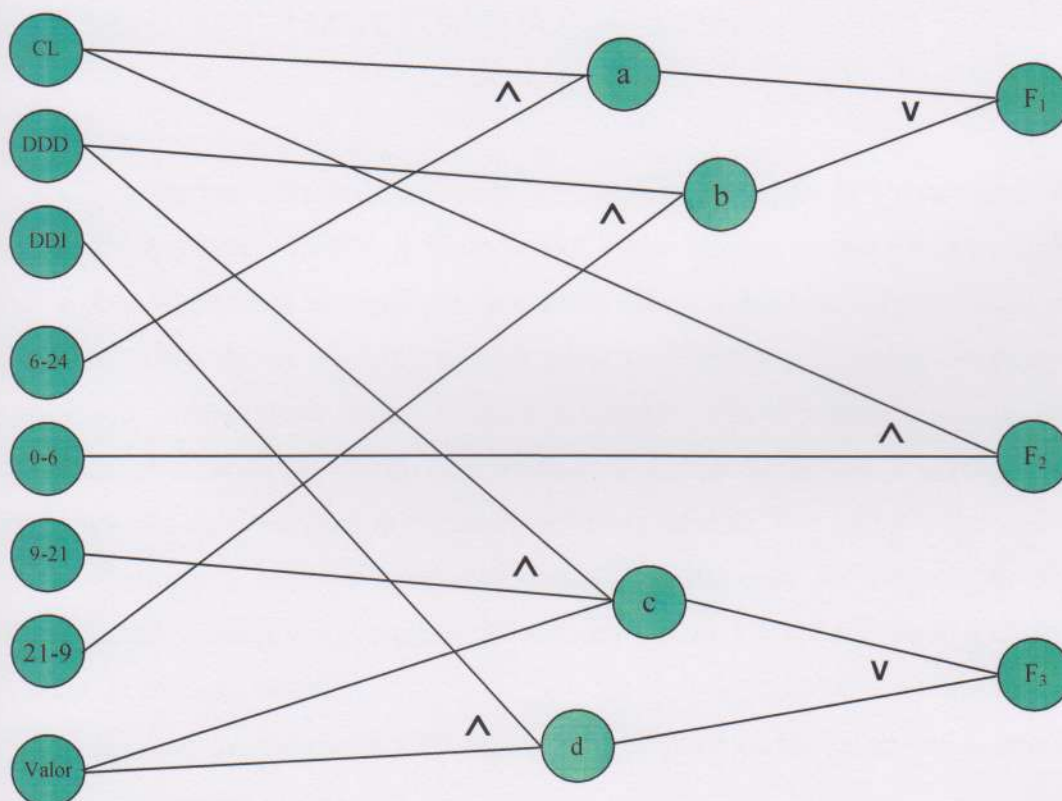


Figura 7 – Grafo de Causa-Efeito
Fonte: Leite, 2008

A partir do grafo se constrói uma tabela de decisão como mostrado no quadro 4.

Causa	Tipo de Chamada	Local (CL)		Interurbana (DDD)		Internacional (DDI)
	Faixa de horário	6-24	0-6	21-9	9-21	Qualquer horário
Efeito	Cálculo da cobrança	F1	F2	F1	F3	F3

Quadro 4 - Tabela de Decisão

Fonte: Dados do Pesquisador, 2008

Dessa tabela são extraídos cinco casos de teste e pode-se verificar se para as condições de entradas correspondentes (causas) o programa realiza os cálculos correspondentes (efeitos).

- **Caso de teste 1:** tipo da chamada = CL, faixa de horário = 6-24, valor localidade = F1
- **Caso de teste 3:** tipo da chamada = DDD, faixa de horário = 21-9, valor localidade = F1
- **Caso de teste 2:** tipo da chamada = CL, faixa de horário = 0-6, valor localidade = F2
- **Caso de teste 4:** tipo da chamada = DDD, faixa de horário = 9-21, valor localidade = F3
- **Caso de teste 5:** tipo da chamada = DDI, faixa de horário = Qualquer horário, valor localidade = F3

3.1 SUPERINTENDÊNCIA DE INFORMÁTICA – UFRN

A Superintendência de Informática – Sinfo é um órgão da Universidade Federal do Rio Grande do Norte – UFRN. A Sinfo possui várias funções dentro e fora da Universidade, como a disseminação de serviços para a comunidade e elaboração da política de informática da UFRN. Além disso, procura ser uma porta de entrada para novas tecnologias para a comunidade universitária, tanto acadêmica quanto administrativa, com o objetivo de disponibilizar recursos de ponta para atender às necessidades dos alunos, funcionários e demais pessoas que usufruem da estrutura da Universidade.

Compete à Sinfo o desenvolvimento e manutenção de sistemas de informações dentro da Universidade como também administrar a Rede UFRN, incluindo a UFRNET que é provedor de acesso à Internet.

Em seu organograma (ver figura 8) a Superintendência de Informática possui a seguinte organização:

- **A Superintendência:** é o órgão executivo exercido pelo Superintendente, designado pelo Reitor, com aprovação do CONSAD, conforme art. 10 parágrafo 3º do Estatuto da UFRN.

- **A Gerência de Administração:** é a área executiva que fornece apoio operacional à Superintendência no que se refere à infra-estrutura, administração e desenvolvimento de pessoal, administração de materiais, patrimônio, finanças e serviços.

- **O Conselho Consultivo:** é o órgão superior com funções consultivas sobre matéria política, administrativa, orçamentária, financeira, patrimonial.

3 IPOJECT - FERRAMENTA DE GERENCIAMENTO DE SOFTWARE

O IProject é uma ferramenta de gerenciamento de software responsável por acompanhar as diversas tarefas executadas pela Diretoria de Sistemas que pertence a Superintendência de Informática/UFRN. Antes de apresentar o IProject é preciso conhecer o contexto onde ele está inserido, descrevendo um pouco sobre os órgãos responsáveis pelo mesmo.

3.1 SUPERINTENDÊNCIA DE INFORMÁTICA – UFRN

A Superintendência de Informática – Sinfo é um órgão da Universidade Federal do Rio Grande do Norte – UFRN. A Sinfo possui várias funções dentro e fora da Universidade, como a disseminação de serviços para a comunidade e elaboração da política de informática da UFRN. Além disso, procura ser uma porta de entrada para novas tecnologias para a comunidade universitária, tanto acadêmica quanto administrativa, com o objetivo de disponibilizar recursos de ponta para atender às necessidades dos alunos, funcionários e demais pessoas que usufruem da estrutura da Universidade.

Compete a Sinfo o desenvolvimento e manutenção de sistemas de informações dentro da Universidade como também administrar a Rede UFRN, incluindo a UFRNET que é provedor de acesso à Internet.

Em seu organograma (ver figura 8) a Superintendência de Informática possui a seguinte organização:

- **A Superintendência:** é o órgão executivo exercido pelo Superintendente, designado pelo Reitor, com aprovação do CONSAD, conforme art. 10 parágrafo 3º do Estatuto da UFRN.
- **A Gerência de Administração:** é a área executiva que fornece apoio operacional à Superintendência no que se refere à infra-estrutura: administração e desenvolvimento de pessoal, administração de materiais, patrimônio, finanças e serviços.
- **O Conselho Consultivo:** é o órgão superior com funções consultivas sobre matéria política, administrativa, orçamentária, financeira, patrimonial.

• **As Comissões Técnicas:** as Comissões Técnicas são instâncias provisórias de trabalho, constituídas de acordo com as necessidades identificadas ou, a partir de recomendação do Conselho Consultivo para realizar atividades específicas.

• **E as Diretorias:** as Diretorias são áreas Técnicas específicas de atividades com funções executivas, exercidas por Diretores, indicados pelo Superintendente de Informática. Existem três diretorias: Sistemas, Redes e Suporte e Informação.

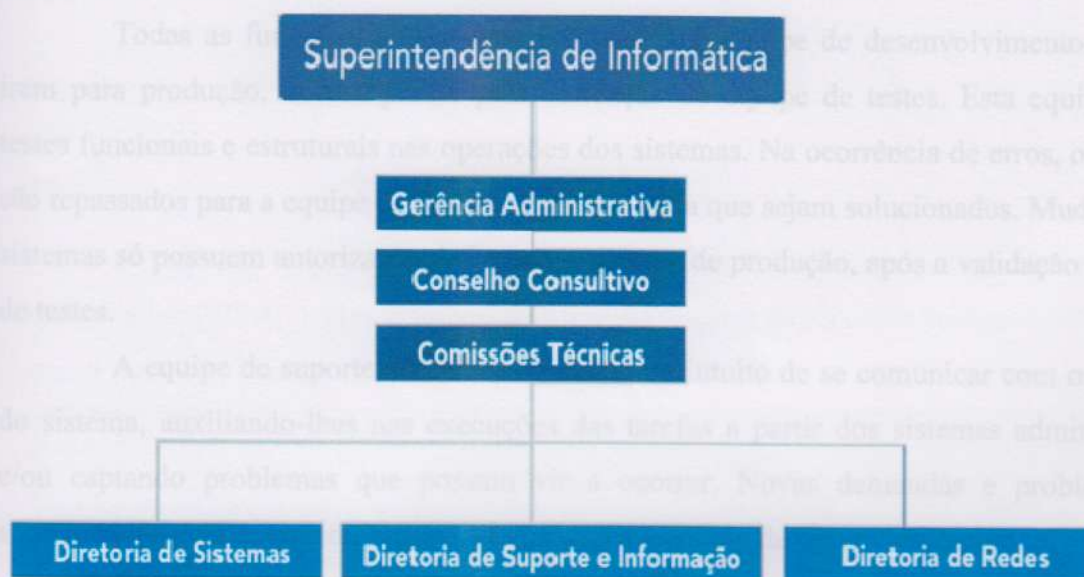


Figura 8 – Organograma da Superintendência de Informática
Fonte: Superintendência de Informática, 2008

3.2 DIRETORIA DE SISTEMAS

A Diretoria de Sistemas é responsável pelo desenvolvimento e manutenção dos sistemas da UFRN. Além do Diretor, esta diretoria é composta por quatro equipes: equipe de requisitos, equipe de suporte ao usuário, equipe de desenvolvimento e equipe de testes.

A equipe de requisitos é responsável por se reunir com usuários dos sistemas para a captação de informações que possam vir a originar novas funcionalidades. Essas informações são denominadas de requisitos dos sistemas e são documentadas. Posteriormente, estas documentações são utilizadas pela equipe de desenvolvimento para o desenvolvimento das novas funcionalidades e pela equipe de testes para auxiliar a validação dessas funcionalidades.

A equipe de desenvolvimento é responsável por desenvolver novas funcionalidades para os sistemas e realizar as suas manutenções. Ela é subdividida em duas equipes, uma que desenvolve e mantém os módulos do sistema administrativo (SIPAC e SIGRH) e outra que desenvolve e mantém os módulos do sistema acadêmico (SIGAA). Cada uma dessas duas equipes possui um líder, que auxilia o diretor de sistemas na gerência de suas atividades. Para as duas subequipes, há um arquiteto de software para manutenção do código fonte da arquitetura existente para auxiliar o desenvolvimento dos sistemas.

Todas as funcionalidades desenvolvidas pela equipe de desenvolvimento, antes de irem para produção, devem passar pela validação da equipe de testes. Esta equipe realiza testes funcionais e estruturais nas operações dos sistemas. Na ocorrência de erros, os mesmos são repassados para a equipe de desenvolvimento para que sejam solucionados. Mudanças nos sistemas só possuem autorização de ir para o sistema de produção, após a validação da equipe de testes.

A equipe de suporte ao usuário trabalha no intuito de se comunicar com os usuários do sistema, auxiliando-lhes nas execuções das tarefas a partir dos sistemas administrativos e/ou captando problemas que possam vir a ocorrer. Novas demandas e problemas não solucionados pela equipe de suporte são passados à equipe de desenvolvimento.

Todo esse fluxo de desenvolvimento adotado pela Diretoria de Sistemas é controlado pelo IProject através das tarefas cadastradas pelas equipes da diretoria.

3.3 O IPROJECT

O Sistema Integrado de Gestão de Projetos – IProject é uma ferramenta desenvolvida pela Diretoria de Sistemas da Superintendência de Informática para facilitar e dinamizar o gerenciamento das tarefas relativas aos projetos desenvolvidos na UFRN.

Com o IProject tem-se um melhor acompanhamento e uma contínua avaliação de todas as tarefas que estão sendo realizadas pelas equipes de requisitos, suporte, desenvolvimento e testes. O IProject prover funcionalidades para acrescentar usuários, solicitar atualização em banco e em produção, solicitar testes e acompanhar os chamados abertos nos sistemas pelos usuários.

Todas essas atividades são registradas no IProject através do cadastramento de tarefas e *logs* que podem ser definidos como anotações das atividades realizadas em cada fase

na execução de uma tarefa. Após a criação de uma tarefa até a sua finalização cada passo realizado para obter sua execução é registrada através de logs. Cada log cadastrado fornece campos para colocar uma descrição, um status, anexar um arquivo e alterar o status da tarefa. Um log pode assumir vários status que informa em que passo de execução se encontra a tarefa. Cada vez que uma tarefa é criada e que um log é adicionado, o IProject se encarrega de enviar automaticamente um e-mail para um ou mais usuários envolvidos na tarefa.

3.4 SISTEMAS GERENCIADOS PELO IPOJECT

Atualmente a UFRN possui projetos de informatização em três grandes áreas: área administrativa (SIPAC), área acadêmica (SIGAA) e a área de recursos humanos (SIGRH). Na figura 9 pode-se visualizar o relacionamento entre os sistemas.

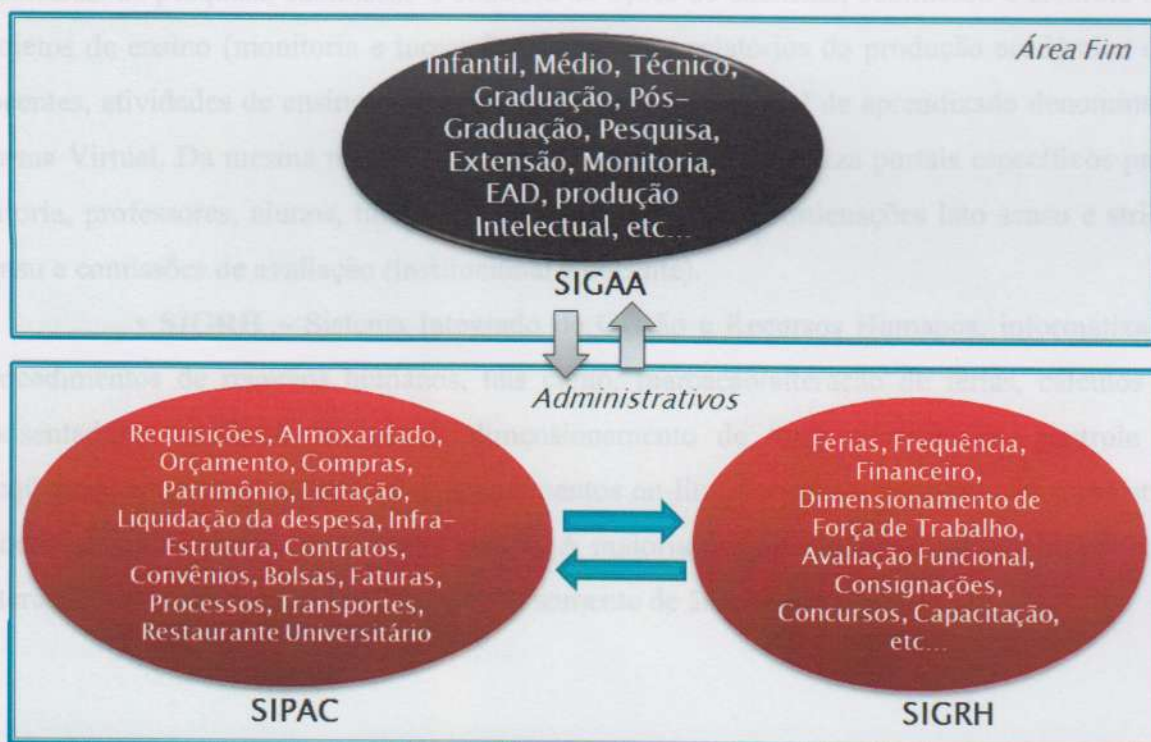


Figura 9 - Relacionamento entre os sistemas da UFRN
Fonte: Galhardo, 2008

Para conhecer melhor os sistemas será apresentada uma breve descrição dos módulos que cada sistema comporta.

- **SIPAC** – Sistema Integrado de Patrimônio, Administração e Contratos, informatiza os fluxos da área administrativa, envolvendo todo o orçamento distribuído no

âmbito interno e das requisições que demandam este orçamento (Material, Passagens, Diárias, Suprimento de Fundos, Auxílio Financeiro, prestações de serviço pessoa física e jurídica, etc.). Informatiza também os almoxarifados (centrais e setoriais), todo o controle patrimonial, as compras e licitações, o controle de atas e pedidos em registros de preços, o acompanhamento de entrega de empenhos (liquidação), o controle de obras e manutenções de bens imóveis, a aquisição de livros pela biblioteca, as faturas de água e energia, o controle dos contratos e convênios celebrados, o fluxo de processos e documentos eletrônicos, o registro e pagamento de bolsistas, o acompanhamento das despesas com automóveis e combustíveis. O SIPAC também disponibiliza portais de informações para os pró-reitores, para a auditoria interna e para a fundação.

- **SIGAA** – Sistema Integrado de Gestão de Atividades Acadêmicas, informatiza os procedimentos da área acadêmica através dos módulos de: graduação, pós-graduação (stricto e lato-sensu), ensino técnico, ensino médio e infantil, submissão e controle de projetos e bolsistas de pesquisa, submissão e controle de ações de extensão, submissão e controle dos projetos de ensino (monitoria e inovações), registro e relatórios da produção acadêmica dos docentes, atividades de ensino a distância e um ambiente virtual de aprendizado denominado Turma Virtual. Da mesma maneira do SIPAC também disponibiliza portais específicos para: reitoria, professores, alunos, tutores de ensino a distância, coordenações lato sensu e stricto sensu e comissões de avaliação (institucional e docente).

- **SIGRH** – Sistema Integrado de Gestão e Recursos Humanos, informatiza os procedimentos de recursos humanos, tais como: marcação/alteração de férias, cálculos de aposentadoria, avaliação funcional, dimensionamento de força de trabalho, controle de frequência, concursos, capacitações, atendimentos on-line, serviços e requerimentos, registros funcionais, relatórios de RH, dentre outros. A maioria das operações possui algum nível de interação com o sistema SIAPE, outras são somente de âmbito interno.

Figura 10 – Diagrama de Casos de Uso do IProject
Fonte: Dados do Pesquisador, 2006

3.5 DIAGRAMA DE CASOS DE USO

Com a definição de que a melhor opção era desenvolver uma ferramenta de gerenciamento própria, logo começaram as reuniões para levantar os requisitos do novo sistema. Como o grupo já estava ciente de suas necessidades e possuía uma vivência prática com outros sistemas de gerenciamento, não foi difícil identificar os casos de uso. A figura 10

retrata o Diagrama de Casos de Uso projetados para o IProject. Ele é mais abstrato e modela o comportamento externo do sistema, tentando apresentar o sistema através de uma perspectiva do usuário.

O Diagrama de Casos de Uso é um tipo de Diagrama da Unified Modeling Language (UML). A UML é um tipo de modelagem usada para especificar, construir e documentar artefatos de sistemas de software. Tendo como objetivo principal descrever qualquer tipo de sistemas (sendo mais comum, criar modelos de sistemas de software), em termos de diagrama orientado a objetos.

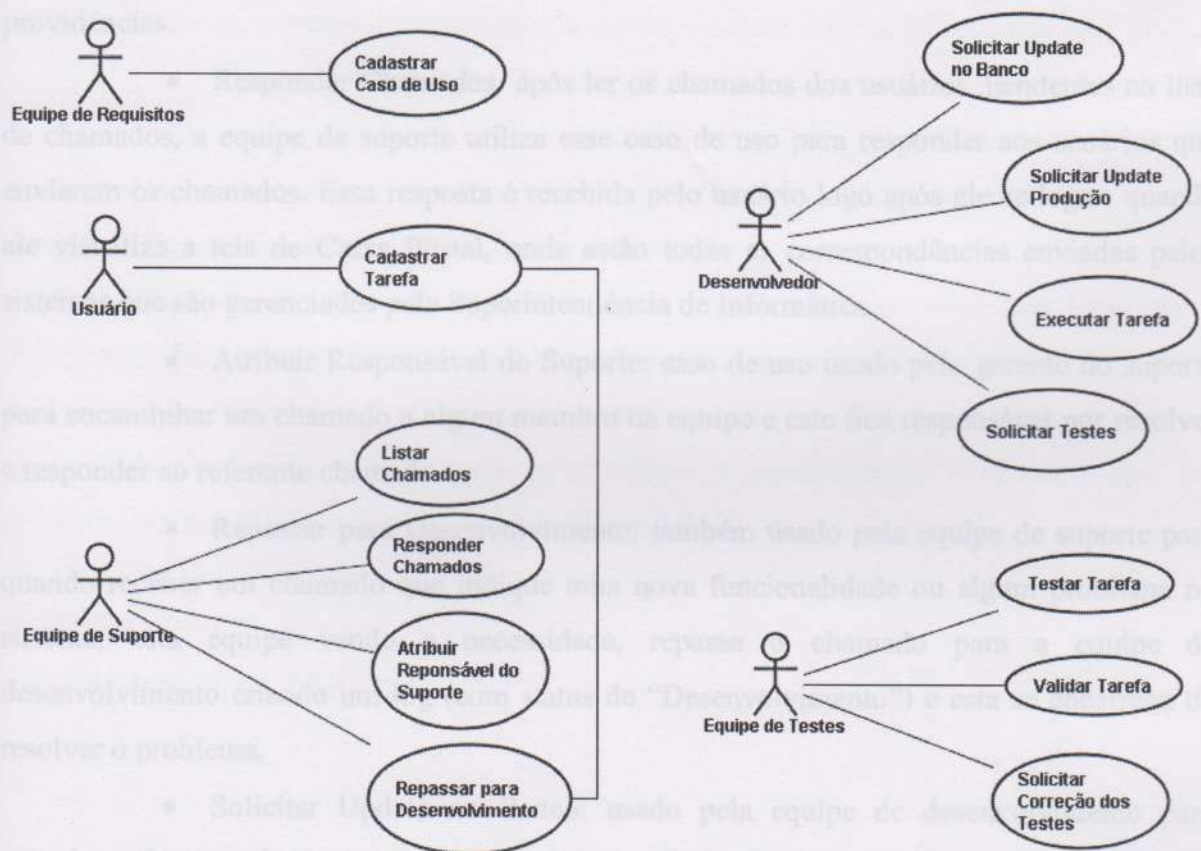


Figura 10 – Diagrama de Casos de Uso do IProject
Fonte: Dados do Pesquisador, 2008

Utilizando o diagrama da figura 10, pode-se conhecer melhor o IProject detalhando cada caso de uso encontrado.

- **Cadastrar Caso de Uso:** é o cadastro dos casos de uso encontrados pela equipe de requisitos quando da análise dos requisitos de um novo sistema.

- **Cadastrar Tarefa:** é o principal caso de uso do IProject, nesse caso de uso todas as tarefas referentes aos projetos são cadastradas e designadas a um responsável da equipe. Qualquer usuário do IProject pode cadastrar uma nova Tarefa.

- **Listar Chamados:** caso de uso que lista todos os chamados enviados pelos usuários e que ainda não foram resolvidos. A abertura de chamados é uma das maneiras que os usuários podem utilizar para entrarem em contato com a equipe de suporte enviando suas dúvidas, sugestões, reclamações ou para reportar problemas que possam vir a ocorrer durante o uso dos sistemas. Os chamados são abertos pelos usuários em qualquer um dos sistemas e a equipe de suporte usa esse caso de uso para ler os chamados enviados e tomar as devidas providências.

- **Responder Chamados:** após ler os chamados dos usuários, pendentes na lista de chamados, a equipe de suporte utiliza esse caso de uso para responder aos usuários que enviaram os chamados. Essa resposta é recebida pelo usuário logo após ele se logar: quando ele visualiza a tela de Caixa Postal, onde estão todas as correspondências enviadas pelos sistemas que são gerenciados pela Superintendência de Informática.

- **Atribuir Responsável do Suporte:** caso de uso usado pelo gerente do suporte para encaminhar um chamado a algum membro da equipe e este fica responsável por resolver e responder ao referente chamado.

- **Repassar para Desenvolvimento:** também usado pela equipe de suporte para quando receber um chamado que indique uma nova funcionalidade ou algum problema no sistema, esta equipe vendo a necessidade, repassa o chamado para a equipe de desenvolvimento criando um log (com status de “Desenvolvimento”) e esta se encarrega de resolver o problema.

- **Solicitar Update no Banco:** usado pela equipe de desenvolvimento para quando, após a conclusão de uma tarefa, necessitar fazer alguma inserção ou alteração na base de dados. O log cadastrado nesse caso de uso fica com status de “Solicita Update no Banco”.

- **Solicitar Update em Produção:** também usado pela equipe de desenvolvimento para quando, após conclusão de uma tarefa, necessitar fazer alguma atualização de código. O log cadastrado nesse caso de uso fica com status de “Solicita Update em Produção”

- **Executar Tarefa:** é resolver a tarefa que lhe foi indicada. Cadastrando na tarefa os logs com observações e alterando a tarefa com a porcentagem da resolução da tarefa já realizada. Durante essa fase os logs são cadastrados com status em “Desenvolvimento”.

- Solicitar Testes: outro caso de uso que a equipe de desenvolvimento utiliza quando a tarefa necessita que seja realizado um teste antes de enviar para produção. No cadastramento do log a equipe de desenvolvimento coloca o status do log como “Solicitação de Testes”.
- Testar Tarefa: após a equipe de desenvolvimento solicitar um teste de uma tarefa, essa tarefa é enviada para a equipe de teste que se encarrega efetuar os testes e caso encontre erros um log de erro é adicionado à tarefa. Nesse log de erro o status da tarefa fica com “Erro de Teste”.
- Validar Tarefa: usado pela equipe de testes, que após testar uma tarefa e não encontrando erros, a equipe adiciona a tarefa um log e coloca o status como “Validada”.
- Solicitar Correção dos Testes: após os testes serem realizados pela equipe de teste, se essa equipe encontrar erros, a tarefa volta para o membro da equipe de desenvolvimento que solicitou o teste para que este faça as devidas correções e em seguida abra um novo log com status de “Solicitação de Testes”.

Além desses casos de uso, o IProject possui algumas telas que facilitam a organização das tarefas pelos usuários. Uma tela muito importante é a tela que lista as tarefas por prioridade, essa tela lista todas as tarefas que foram designadas para o usuário, organizando de acordo com o nível de prioridade e possibilitando o usuário fazer um planejamento diário, semanal ou mensal de suas tarefas.

Figura 11 – Diagrama de Classes do IProject
Fonte: Dados da Pesquisa, 2008

3.6 DIAGRAMA DE CLASSES

3.7 DIAGRAMA DE ATIVIDADES

Para melhor compreender o funcionamento do IProject, um Diagrama de Classes é apresentado na figura 11. Esse diagrama apresenta uma visão estática de como as classes estão organizadas. É outro tipo de diagrama da UML.

O Diagrama de Atividades é um tipo de diagrama da UML que modela os aspectos dinâmicos do sistema e as etapas sequenciais. Ele tenta acompanhar as mudanças de um estado de ação sofrido por um objeto dentro de um determinado processo.

A figura 12 mostra o Diagrama de Atividades referente ao sistema IProject. Nesse diagrama pode-se acompanhar o fluxo das atividades realizadas por esta aplicação.

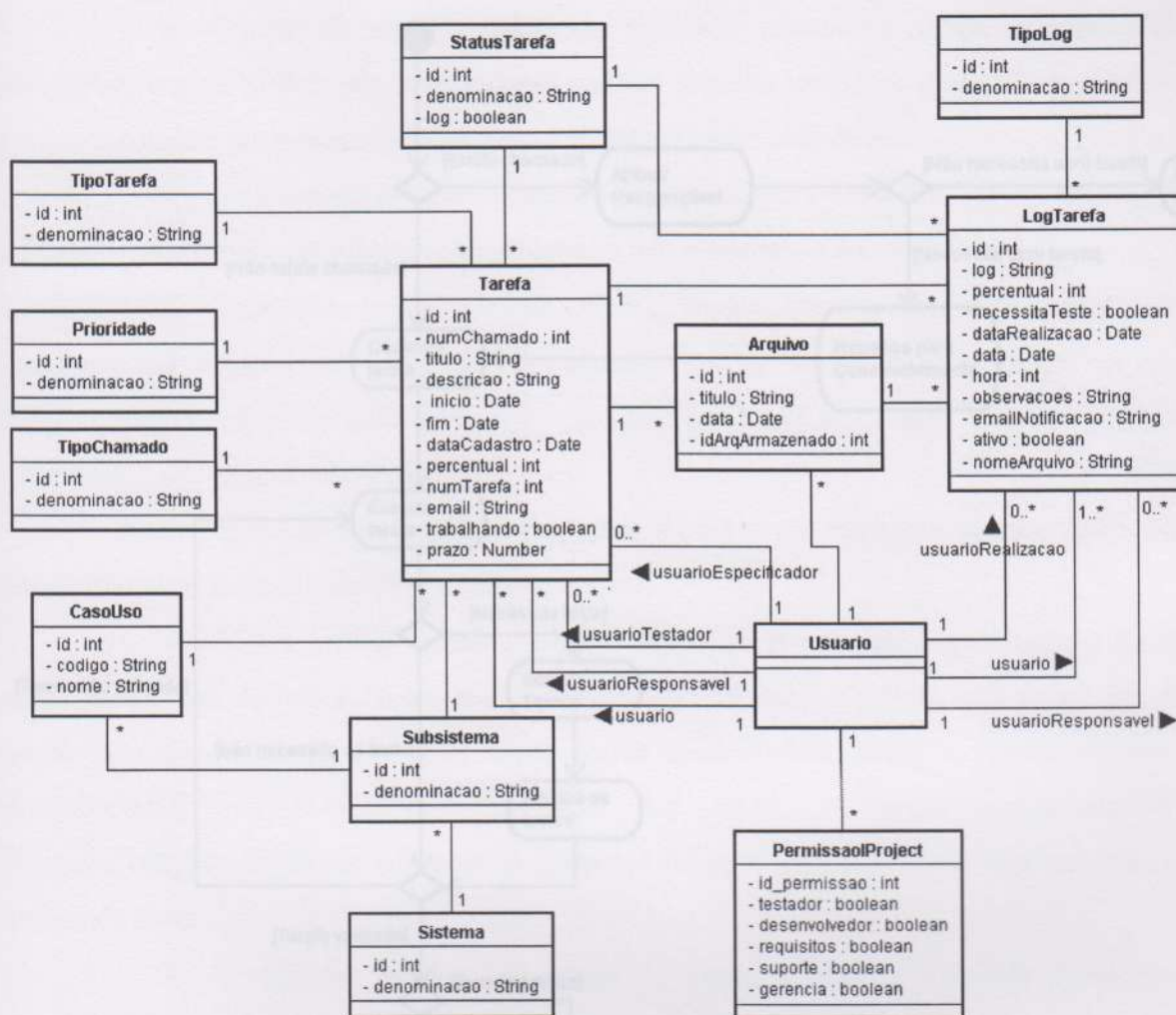


Figura 11 – Diagrama de Classes do IProject
Fonte: Dados do Pesquisador, 2008

3.7 DIAGRAMA DE ATIVIDADES

Através deste diagrama é possível observar o fluxo das atividades realizadas pelo IProject. O Diagrama de Atividades é um tipo de diagrama da UML que modela os aspectos dinâmicos do sistema e as etapas seqüenciais. Ele tenta acompanhar as mudanças de um estado de ação sofrido por um objeto dentro de um determinado processo.

A figura 12 mostra o Diagrama de Atividades referente ao sistema IProject. Nesse diagrama pode-se acompanhar o fluxo das atividades realizadas por esta aplicação.

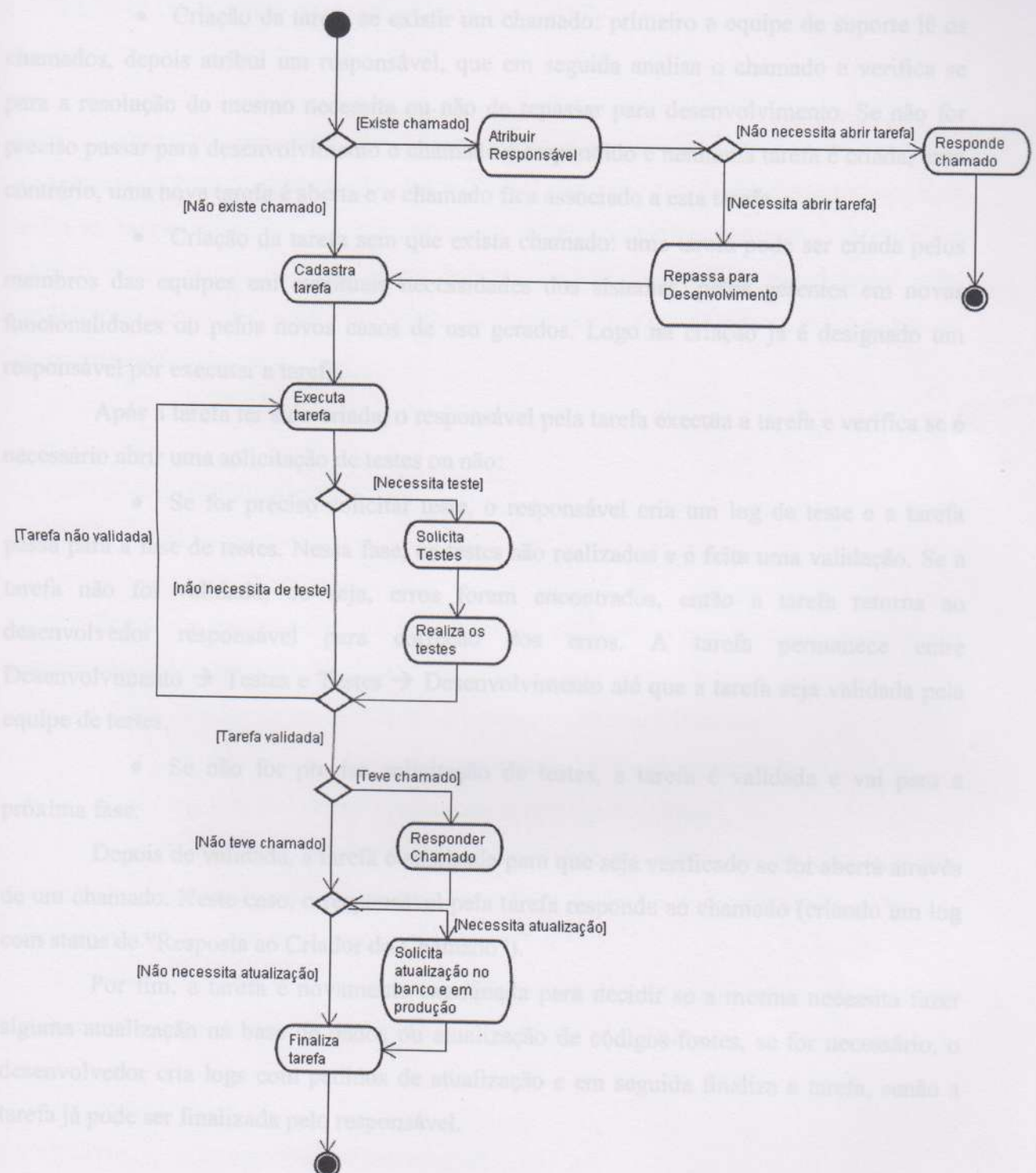


Figura 12 - Diagrama de Atividades do IProject
 Fonte: Dados do Pesquisador, 2008

Para entender melhor o fluxo realizado no diagrama de atividades da figura 12 algumas explicações são necessárias. O início do diagrama mostra que existem dois modos de criar uma tarefa:

4.1 TESTES DE CONSISTÊNCIA EM FORMULÁRIOS

- Criação da tarefa se existir um chamado: primeiro a equipe de suporte lê os chamados, depois atribui um responsável, que em seguida analisa o chamado e verifica se para a resolução do mesmo necessita ou não de repassar para desenvolvimento. Se não for preciso passar para desenvolvimento o chamado é respondido e nenhuma tarefa é criada, caso contrário, uma nova tarefa é aberta e o chamado fica associado a esta tarefa.

- Criação da tarefa sem que exista chamado: uma tarefa pode ser criada pelos membros das equipes em eventuais necessidades dos sistemas, pelos gerentes em novas funcionalidades ou pelos novos casos de uso gerados. Logo na criação já é designado um responsável por executar a tarefa.

Após a tarefa ter sido criada, o responsável pela tarefa executa a tarefa e verifica se é necessário abrir uma solicitação de testes ou não:

- Se for preciso solicitar teste, o responsável cria um log de teste e a tarefa passa para a fase de testes. Nessa fase, os testes são realizados e é feita uma validação. Se a tarefa não foi validada, ou seja, erros foram encontrados, então a tarefa retorna ao desenvolvedor responsável para correção dos erros. A tarefa permanece entre Desenvolvimento → Testes e Testes → Desenvolvimento até que a tarefa seja validada pela equipe de testes.

- Se não for preciso solicitação de testes, a tarefa é validada e vai para a próxima fase.

Depois de validada, a tarefa é analisada para que seja verificado se foi aberta através de um chamado. Neste caso, o responsável pela tarefa responde ao chamado (criando um log com status de “Resposta ao Criador do Chamado”).

Por fim, a tarefa é novamente examinada para decidir se a mesma necessita fazer alguma atualização na base de dados ou atualização de códigos-fontes, se for necessário, o desenvolvedor cria logs com pedidos de atualização e em seguida finaliza a tarefa, senão a tarefa já pode ser finalizada pelo responsável.

4.2 TESTES DE CONSISTÊNCIA EM FORMULÁRIOS

São testes realizados em campos de formulários de cadastro. Cada campo é testado de acordo com o tipo de dado que está apto a receber.

- Submissão de formulário sem preenchimento dos campos: caso de teste para verificar se o sistema trata os campos obrigatórios, mostrando as mensagens de advertências.

4 TESTES FUNCIONAIS NOS SISTEMAS DA UFRN

A equipe de teste, além de outras atividades, é a responsável pelos testes funcionais nos sistemas da UFRN - SIPAC, SIGAA e SIGRH, seguindo um determinado roteiro. Esse roteiro se baseia em certos padrões que foram definidos pelos gerentes dos sistemas em conjunto com a equipe de testes e em algumas técnicas de testes já mencionadas neste trabalho e que foram adaptadas a instituição. Conheça detalhes dos principais testes realizados por essa equipe:

4.1 TESTES DE USABILIDADE

Testa se o usuário identifica claramente as informações que precisa e as ações que deseja realizar ao lidar com a interface.

- Analisar se um usuário final consegue entender a interface.
- Se os termos técnicos usados possuem significado explicitado.
- Os links têm nomes condizentes com a sua finalidade?
- Testar se os ícones colocados dão uma boa indicação de qual link será referenciado. Analisar se seguem a tabela de padrões de ícones e se possuem uma legenda informando o que o ícone se propõe a fazer.
- Analisar se os textos e as mensagens emitidas em todo sistema estão escritos corretamente (possíveis erros de ortografia) e entendíveis ao usuário.

4.2 TESTES DE CONSISTÊNCIA EM FORMULÁRIOS

São testes realizados em campos de formulários de cadastros. Cada campo é testado de acordo com o tipo de dado que está apto a receber.

- Submissão de formulário sem preenchimento dos campos: caso de teste para verificar se o sistema trata os campos obrigatórios, mostrando as mensagens de advertências.

Inclusive verificando se o sistema ao aceitar espaços em branco detecta que o campo está vazio.

Por exemplo: ver se ao deixar um campo obrigatório sem preenchimento, se está sendo emitida uma mensagem de advertência.

- Verificar campos com máscara: caso de teste que analisa se os campos que necessitam de máscaras estão mascarados devidamente corretos.

Por exemplo: o campo CPF deve ser preenchido de acordo com a seguinte máscara NNN.NNN.NNN-NN.

- Submissão de dados inválidos: caso de teste que testa se os campos estão preparados para receberem ou não valores inválidos.

Por exemplo: os campos:

- Numéricos ou de Valores: não devem aceitar caracteres e se aceitarem devem fazer o tratamento correto;

- Data: não deve aceitar caracteres, nem datas inválidas como 31/02/2008 ou 20/15/2008, etc.;

- CPF/CNPJ: não deve aceitar CPFs e CNPJs inválidos ou que não existam.

- Com Ajax: se faz o tratamento ao submeter com dados errados;

- Verificar a quantidade de caracteres aceitos pelos campos: ver se os campos estão aceitando um número de caracteres maior do que o tamanho fixado na tabela do banco.

Por exemplo: se um campo “Nome” na tabela do banco possui 60 caracteres, então no formulário da página esse campo deve estar aceitando apenas os 60 caracteres.

- Verificar o valor limite dos campos: ver qual o valor máximo e mínimo permitido para determinado campo. Testar se ao inserir um valor fora do limite, está sendo feito o tratamento adequado.

Por exemplo: um campo que aceita valores maiores que 0, se ao colocar um valor “-1”, ver se está sendo tratado corretamente.

- Verificar se os campos textos estão aceitando caracteres especiais e acentos.

- Ver se os botões rotulados “Voltar”, usado para retornar a página imediatamente anterior e “Cancelar”, usado para sair de uma operação antes de concluí-la, estão funcionando corretamente.

- Testar se consigo retornar (sem usar o botão “Voltar” do navegador) para a tela anterior, se for o caso.

4.3 TESTES DE TEMPO DE RESPOSTA E INTEROPERABILIDADE

São testes que verificam se os acessos as páginas estão em níveis aceitáveis de rapidez e compatibilidade com vários browsers e sistemas operacionais.

- Testar se as páginas estão sendo rapidamente acessadas, ou seja, se o tempo de resposta é baixo.
- Ver se as páginas estão funcionando corretamente para os principais browsers dos principais sistemas operacionais.
- O site é acessível na resolução de 800 x 600?

4.4 PADRONIZAÇÃO DE LAYOUT

Testes que verificam se as páginas estão seguindo os padrões estéticos definidos no início do desenvolvimento.

- Verificar se o layout das páginas está seguindo o padrão que foi determinado.
- O sistema possui uma consistência de layout, isto é, claramente se reconhece o padrão de interface?
- Testa se as páginas usam o esquema padrão de cores definidos no sistema.

4.5 TESTES DE NAVEGAÇÃO

Esses testes verificam se o fluxo de navegação das páginas está em conformidade com os links apresentados e se segue o padrão adotado.

- Ver se os botões rotulados “Voltar”, usado para retornar a página imediatamente anterior e “Cancelar”, usado para sair de uma operação antes de concluí-la, estão funcionando corretamente;
- Testar se consigo retornar (sem usar o botão “Voltar” do navegador) para a tela anterior, se for o caso.

- Testar se consegue cancelar facilmente uma operação.
- Ver se o botão “Cancelar” está pedindo uma confirmação antes de sair da operação em definitivo.
- Todo o fluxo do caso de uso funciona corretamente se o usuário voltar usando o “Voltar” do navegador?
- Fazer uma operação (incluir, alterar ou remover), voltar pelo browser e realizá-la novamente; nesse caso, após efetuar a operação pela segunda vez é para o sistema mostrar a mensagem de “Solicitação já processada”.
- Ver se a ordem dos botões – Voltar, Cancelar, Cadastrar, Continuar, Adicionar etc. – estão seguindo os padrões dos sistemas.

Por exemplo:

- <<Voltar Cancelar Continuar>>
- Gravar <<Voltar Cancelar

4.6 TESTES DE INTEGRIDADE DOS DADOS

Testes realizados com o uso do banco de dados para verificar se os dados manipulados pelo sistema estão coerentes com os dados armazenados nos Banco de Dados.

- Testar se as operações de inserção, alteração e remoção realizadas nos sistemas estão sendo feitas realmente na base de dados.
- Verificar se os dados dos relatórios e das consultas estão sendo trazidos corretamente da base de dados.

4.7 TESTES DOS CASOS DE USO

Analisar se as páginas estão realmente realizando os fluxos completos dos casos de uso e se eles estão em conformidade com os requisitos funcionais analisados pela equipe de requisitos.

5 PROJETO DE INCORPORAÇÃO DOS TESTES FUNCIONAIS AO IPOJECT

externamente e os relacionamentos entre eles.

Um padrão de arquitetura de software bastante usado no desenvolvimento de sistemas.

Quando o IPOject foi planejado ainda não havia a tendência de se usar essa ferramenta para gerenciar as atividades de testes. Para sanar esta deficiência, foram implementadas novas funcionalidades ao IPOject para que ele pudesse atender as necessidades referentes à realização dos testes funcionais pela equipe de testes. Esses aprimoramentos foram realizados principalmente no fluxo existente entre a equipe de desenvolvimento e a equipe de testes. O intuito é de melhorar e facilitar o diálogo entre os desenvolvedores quando solicitam testes e pelos testadores quando executam os testes e reportam erros.

As mudanças acontecem em três casos de uso do IPOject: o caso de uso Solicitar Testes e Erro de Testes, que são usados pela equipe de desenvolvimento e o caso de uso Testes Realizados, usado pela equipe de testes.

5.1 TECNOLOGIAS UTILIZADAS

No projeto do IPOject aplicou-se uma arquitetura de desenvolvimento de software em camadas, baseado na plataforma de desenvolvimento Java EE (Java Enterprise Edition) e foi utilizado um conjunto de tecnologias apropriadas para o desenvolvimento de sistemas web. O IPOject usa a mesma arquitetura dos sistemas por ele gerenciados. Para um melhor entendimento, faz-se necessário apresentar alguns conceitos referentes ao desenvolvimento em camadas e as tecnologias empregadas.

5.1.1 ARQUITETURA DE SOFTWARE EM CAMADAS

Os softwares estão cada vez maiores, mais complexos e sofrendo constantes modificações. Na tentativa de reduzir o tempo e o custo de desenvolvimento e manutenção dos sistemas utilizamos uma **Arquitetura de Software**. A arquitetura de software define a

estrutura do software, que compreende os componentes com suas propriedades visíveis externamente e os relacionamentos entre eles.

Um padrão de arquitetura de software bastante usado no desenvolvimento de sistemas é a **Arquitetura em Camadas**. Essa arquitetura divide um sistema considerado grande em camadas, o que diminui consideravelmente a sua complexidade e facilita a sua manutenção.

Segundo Phillip Calçado (CALÇADO, 2006) em seu artigo sobre “Arquitetura de Camadas em Java EE”, as camadas “são divisões lógicas de componentes agrupados por responsabilidades em comum” e a arquitetura em camadas “é um Padrão Arquitetural que ajuda na tarefa de separar responsabilidades, promovendo baixo acoplamento e alta coesão em um sistema.”. Onde, segundo ele, coesão e acoplamento são métricas básicas em um módulo de software, sendo que coesão “é a medida em que os componetes (classes, pacotes etc.) de um módulo estão relacionados, se eles têm responsabilidades em comum ou se foram apenas agrupados por acaso.” e acoplamento “diz respeito à independência entre componentes, a medida do impacto que a alteração de implementação de um componente tem sobre os outros.”.

Com a arquitetura em camadas conseguimos: reduzir a complexidade, reduzir dependência e acoplamento, favorecer a coesão, promover reusabilidade, entre outras vantagens. Em relação as desvantagens podemos citar a limitação da tecnologia e poder complicar sistemas simples.

Num processo de desenvolvimento podemos utilizar um padrão com as seguintes camadas:

- Camada de Apresentação – responsável pela exibição do conteúdo do modelo e interação com os usuários.
- Camada de Negócio ou Domínio – representa os dados corporativos e as regras de negócio que controlam o acesso e o processamento desses dados.
- Camada de Aplicação – monitora como os objetos de negócio são utilizados pela camada de apresentação, ligando essas duas camadas.
- Camada de Persistência – tem a função de armazenar os objetos do domínio.

A figura 13 mostra como essas camadas estão estruturadas.

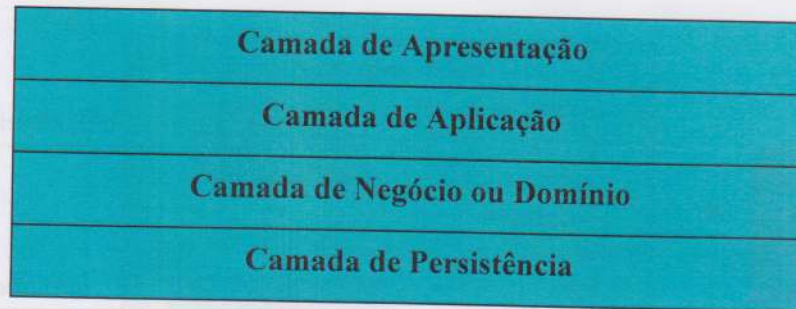


Figura 13 – Arquitetura de software em camadas
Fonte: Galhardo, 2008

5.1.2 ARQUITETURA MODEL-VIEW-CONTROLLER (MVC)

A arquitetura MVC (Modelo – Visualização – Controle) é um padrão que divide a funcionalidade envolvida na manutenção e apresentação dos dados de uma aplicação. Tem a finalidade de separar a lógica da aplicação (Modelo), da interface do usuário (Visualização) e do fluxo da aplicação (Controle). Onde:

- Modelo - representa os dados usados pela aplicação e as operações de negócio associadas a eles;
- Visualização - é responsável pelos detalhes da implementação da interface do usuário;
- Controle - contém as informações sobre o fluxo de programa/transição da interface e captura os eventos gerados pela camada de visualização.

O padrão MVC surgiu para resolver um problema na camada de apresentação onde a lógica de negócio ficava fortemente acoplada com a interface. Com a utilização desse padrão há uma diminuição do acoplamento e as alterações realizadas na interface não afetam os dados da lógica de negócio, permitindo que a mesma lógica de negócios possa ser acessada e visualizada por várias interfaces.

A figura 14 mostra uma arquitetura MVC.

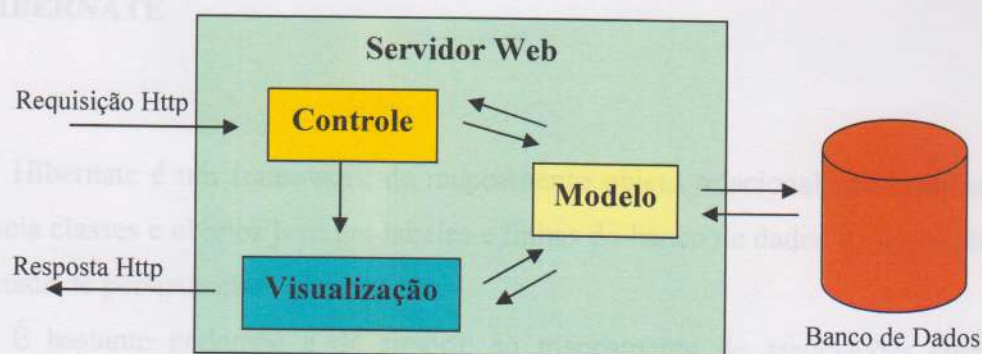


Figura 14 – Arquitetura MVC
Fonte: Telemaco, 2007

5.1.3 JAVA SERVER FACES (JSF)

É um framework MVC para construção de interfaces de usuário para aplicações da web. É responsável pela interação com dispositivos clientes e fornece as ferramentas para criar a apresentação visual, a lógica da aplicação e a lógica de negócio. JSF possui uma clara separação entre as camadas de modelo, visualização e controle. Essa separação torna a camada de visualização independente, podendo serem geradas diversas saídas além do padrão HTML.

É um framework de desenvolvimento web orientado a eventos e oferece vários serviços como: conversão automática de dados, validação e tratamento de erros, permite criar facilmente aplicações internacionalizadas etc. Seu escopo se restringe na camada de apresentação.

5.1.4 ENTERPRISE JAVA BEANS (EJB)

EJB define uma arquitetura para o desenvolvimento rápido e simplificado de aplicações Java baseado em componentes distribuídos e transacionais que são gerenciados pelo container EJB. É a principal tecnologia da plataforma Java EE. O EJB está na camada de negócio.

5.1.5 HIBERNATE

Hibernate é um framework de mapeamento objeto relacional para aplicações Java. Ele mapeia classes e objetos java em tabelas e linhas do banco de dados e vice-versa. É usado pela camada de persistência

É bastante poderoso e dá suporte ao mapeamento de associações entre objetos, herança, polimorfismo, composição e coleções. Também disponibiliza um poderoso mecanismo de consulta de dados, permitindo uma redução considerável no tempo de desenvolvimento da aplicação.

Com o hibernate conseguimos maior produtividade, menor impacto após as mudanças e independência de fornecedor, uma vez que usa dialetos SQL.

5.1.6 POSTGRESQL

PostgreSQL é um Sistema Gerenciador de Banco de Dados (SGBD) objeto relacional, extremamente robusto, confiável, extremamente flexível e rico em recursos. Ele é considerado objeto relacional, pois, além de ser um SGBD relacional possui algumas características de orientação a objetos. Também faz parte da camada de persistência.

5.2 NOVO DIAGRAMA DE CLASSES DO IPOJECT

Depois de analisar as alterações a serem realizadas no IProject, foi desenvolvido um novo diagrama de classes que contempla a atividade de testes (ver figura 15).

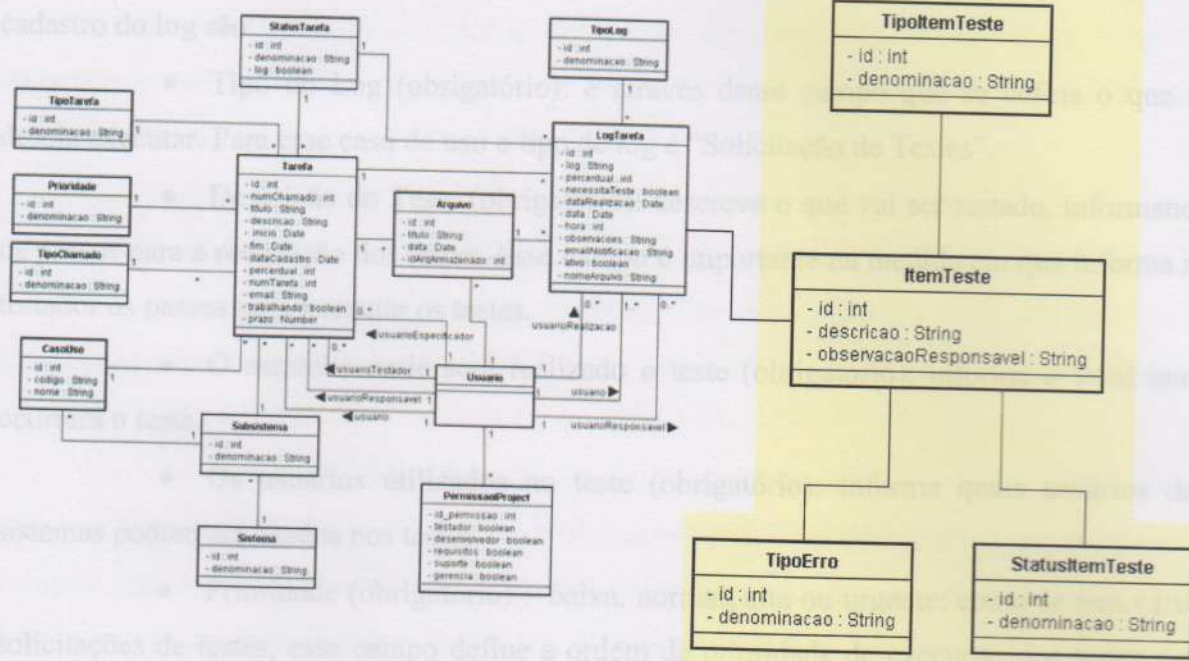


Figura 15 – Novo Diagrama de Classe do IProject
Fonte: Dados do Pesquisador, 2008

5.3 CASO DE USO ‘SOLICITAR TESTES’

Na versão inicial do IProject esse caso de uso foi apenas adaptado aos testes, o que deixava muito a desejar, pois a solicitação não era bem definida pela equipe de desenvolvimento e causava alguns transtornos na realização dos testes. Além disso, os relatórios emitidos não conseguiam fornecer as informações desejadas pelos gerentes.

Para contornar essa situação, nesse novo projeto foram adicionados novos dados que melhoraram consideravelmente o entendimento da solicitação pela equipe de testes. O caso de uso começa com a Solicitação de Testes realizada pela equipe de desenvolvimento após execução da tarefa. Essa solicitação é feita cadastrando um log na tarefa e preenchendo o tipo do log para “Solicitação de Testes”. Quando esse tipo de log é escolhido campos adicionais aparecem ao log com as novas informações referentes à atividade de testes. Esses campos fornecem informações que dão subsídios para a equipe de testes estruturar o caso de teste e em seguida realizar o teste propriamente dito. As novas informações acrescentadas foram baseadas em uma estrutura de caso de teste e adaptada ao fluxo de desenvolvimento da instituição. De modo que os testes não sejam um gargalo e que a equipe de testes possua

informações necessárias para realizar um bom teste. Os campos para preenchimento no cadastro do log são:

• Tipo do Log (obrigatório): é através desse campo que se define o que se deseja executar. Para esse caso de uso o tipo de log é “Solicitação de Testes”.

• Descrição do Teste (obrigatório): descreve o que vai ser testado, informando os passos para a realização dos testes. Esse campo é importante na medida em que informa ao testador os passos para executar os testes.

• O caminho onde será realizado o teste (obrigatório): informa o local onde ocorrerá o teste.

• Os usuários utilizados no teste (obrigatório): informa quais usuários dos sistemas podem ser usados nos testes.

• Prioridade (obrigatório) – baixa, normal, alta ou urgente: como se tem várias solicitações de testes, esse campo define a ordem de prioridade de execução dos testes pela equipe de testes.

• Validações esperadas: informa possíveis validações a serem feitas pela equipe de testes para determinado teste.

• Ambiente e ferramentas a serem utilizados: informações sobre o ambiente, softwares e outras configurações necessárias para a realização dos testes (ex.: Sistema Operacional, browsers, outros softwares...)

Depois que a solicitação é efetuada, existe uma tela que lista os testes pendentes, isto é, são tarefas que solicitaram testes e ainda não foram testadas. Essa lista é constantemente visitada pela equipe de testes a fim de que os testes possam ser realizados logo após serem feitas as solicitações.

5.4 CASO DE USO ‘TESTE REALIZADO’

Caso de uso utilizado pela equipe de testes, para quando os testes forem finalizados e, em caso de teste bem sucedido (quando forem encontrados erros) essa equipe possa adicionar os erros ocorridos ao log da tarefa. Para isso é cadastrado um log com o tipo “Teste Realizado” onde são cadastrados os itens com os resultados dos testes (erros, sugestões e observações). Para cada item encontrado no teste serão preenchidos os seguintes campos:

- **Tipo do Item de Teste (obrigatório)** – se é um Erro, Sugestão, Observação ou Erro Persiste: define se após os testes foi encontrado erro, sugestão ou observação. O tipo de item “Erro Persiste” é usado quando há um segundo teste e o erro encontrado anteriormente continua existindo.

- **Tipo do Erro (obrigatório)**: só aparece para preenchimento se o tipo do Tipo do Item de Teste for “Erro”. Esse campo informa qual o tipo de erro encontrado (ex.: Campo numérico aceitando caracteres, Erro de NullPointerException, Campo Data não formatada, Erros ortográficos, Link duplicado, Link Errado, Erro de página não encontrada, Layout fora do padrão etc.).

- **Descrição (obrigatório)**: descreve detalhadamente o erro, sugestão ou observação para que a equipe de desenvolvimento possa visualizar e tomar as decisões cabíveis (em caso de um erro, essa equipe deverá corrigi-lo). Ex.: se o tipo de resultado for Erro esse campo deve conter informações de como ocorreu o erro, em que local, em que situações, que usuário logado, em que ambiente etc.

Quando o resultado dos testes é cadastrado, um novo status é definido para a tarefa. Se a tarefa possuir itens do tipo ‘Erro’, ‘Sugestão’ ou ‘Erro Persiste’ o status da tarefa fica como ‘Erro de Teste’ e se a tarefa possuir apenas itens do tipo ‘Observação’ o status da tarefa fica como ‘Validada’. Depois de encerrado o cadastro do log, a tarefa aparece automaticamente na lista de tarefas pendentes do responsável com o status correspondente ao que foi cadastrado. O responsável ao abrir o log verá uma listagem com os resultados dos testes. No caso do status da tarefa está como ‘Erro de Teste’, o responsável irá analisar e/ou resolver cada item desse resultado e em seguida preencher os seguintes campos:

- **Status do Item de Teste (obrigatório)** – Erro, Normal, Corrigido, Repassado, Aceita, Recusada e Em Análise: nesse campo o responsável poderá informar o estado de cada Tipo de Item de Teste.

- Se o Tipo do Item for Erro, o status informa se um erro foi corrigido ou não, se é um funcionamento normal do sistema (não é erro) ou se o erro foi repassado em outra tarefa para outro responsável.

- Se o Tipo do Item for ‘Sugestão’, o status informa se a sugestão foi aceita, recusada ou está sendo analisada.

- Se o Tipo do Item for ‘Observação’, não é necessário informar o status.

- **Observações do Responsável**: são informações inseridas pelo responsável da tarefa para justificar sua posição quando ao tipo de Status escolhido.

6 CON Finalizadas as análises e/ou correções dos resultados, o responsável pela tarefa faz uma nova solicitação de testes. Voltando para testes, as correções e/ou observações colocadas pelo responsável são analisadas, novos testes poderão ser feitos e a equipe poderá ou não validar a tarefa. Caso a tarefa não seja validada, será necessário reenviá-la para o responsável da tarefa com os novos resultados encontrados. O responsável pela tarefa só poderá passar para o próximo passo do fluxo após a tarefa ter sido validada. Com a garantia de que as correções e as novas funcionalidades só entrem em produção depois de validadas pela equipe de testes, os sistemas da UFRN podem ter relativamente um bom índice de qualidade.

Os erros frequentes, o mau funcionamento, ou a inadequação aos requisitos são sempre notados.

A atividade de testes, quando bem realizada, torna-se uma forma de avaliar e agregar qualidade ao produto, reduzir custos e retrabalho, melhorando a imagem da empresa e ampliando sua capacidade competitiva.

Por esta razão, o processo de testes deve ser tratado como mais um processo de software, devendo estar integrado ao desenvolvimento. Os esforços de planejamento e controle da execução de testes devem ser iniciados juntamente com o projeto, para propiciar maior controle e monitoramento das atividades de testes de acordo com a qualidade de software que se pretende gerar.

Foi visto também como o uso de uma ferramenta de gerenciamento de tarefas, o JProject, pode facilitar o trabalho dos gerentes no processo de desenvolvimento. Auxiliando tanto na parte de organizar as tarefas como nos relatórios gerenciais emitidos. Com as novas tarefas de testes adicionadas ao JProject, pôde-se notar um grande avanço na fluidez das tarefas de testes e uma melhor qualidade no software desenvolvido.

No caso da UFRN, conseguiu-se não apenas informatizar boa parte dos seus sistemas, mas também aumentar a garantia e a qualidade do software desenvolvido pela mesma, que embora pública, tem fins lucrativos e sem querer disputar o mercado, se preocupa com o produto que está sendo desenvolvido, procurando atender um público considerável e exigente de Funcionários, Docentes e Alunos que compõe a Universidade Federal do Rio Grande do Norte.

6 CONCLUSÕES

Com este trabalho procurou-se retratar como a UFRN instituiu os testes em seus softwares, como também a importância de se testar um software, mostrando tantos os aspectos da qualidade exigida pelo cliente como do aumento dos custos quando um erro não é identificado logo no início do desenvolvimento. Embora seja difícil medir e definir um software como sendo de boa qualidade, é fácil identificar um software de má qualidade. Os erros frequentes, o mau funcionamento, ou a inadequação aos requisitos são sempre notados.

A atividade de testes, quando bem realizada, torna-se uma forma de avaliar e agregar qualidade ao produto, reduzir custos e retrabalho, melhorando a imagem da empresa e ampliando sua capacidade competitiva.

Por estas razões, o processo de testes deve ser tratado como mais um processo de software, devendo estar integrado ao desenvolvimento. Os esforços de planejamento e controle da execução de testes devem ser iniciados juntamente com o projeto, para propiciar realimentação e para que a atividade de testes esteja de acordo com a qualidade do software que se pretende gerar.

Foi visto também como o uso de uma ferramenta de gerenciamento de tarefas, o IProject, pode facilitar o trabalho dos gerentes no processo de desenvolvimento. Auxiliando tanto na parte de organizar as tarefas como nos relatórios gerenciais emitidos. Com as novas tarefas de testes adicionadas ao IProject, pôde-se notar um grande avanço no fluxo das tarefas de testes e uma melhor qualidade no software desenvolvido.

No caso da UFRN, conseguiu-se não apenas informatizar boa parte dos seus sistemas, mas também aumentar a garantia e a qualidade do software desenvolvido pela mesma, que embora pública, sem fins lucrativos e sem querer disputar o mercado, se preocupa com o produto que está sendo desenvolvido, procurando atender um público considerável e exigente de Funcionários, Docentes e Alunos que compõe a Universidade Federal do Rio Grande do Norte.

REFERÊNCIAS

- BLAZUZ, D. O. **PostgreSQL: introdução e histórico**. Disponível em: <http://www.postgresql.org.br/Introdu%C3%A7%C3%A3o_e_hist%C3%B3rico>. Acesso em: 11 nov. 2008
- CALÇADO, P. Arquitetura de camadas em Java EE. **Revista Mundo Java**. 15.ed. jan. 2006. Disponível em: <<http://www.mundojava.com.br/NovoSite/15materia capa.shtml>>. Acesso em: 10 nov. 2008.
- FAGUNDES, M. **Arquitetura de software**. Disponível em: <<http://www.ufpa.br/cdesouza/teaching/es/Arquitetura-de-Software.pdf>>. Acesso em: 10 nov. 2008
- FERREIRA, A. R. **Testabilidade de software**. Disponível em: <http://www.garcia.pro.br/orientacoes/Ardalla_TCC2.pdf>. Acesso em: 16 out. 2008.
- GALHARDO, R. **Sistemas institucionais: guia de desenvolvimento da Superintendência de Informática – UFRN**. Natal, 2008
- GEARY, David e HORSTMANN, Cay. **Core Java Server Faces: o guia autorizado**. Rio de Janeiro: Alta Books, 2005.
- LEITE, J. C. **Engenharia de software**. Disponível em: <<http://www.dimap.ufrn.br/~jair/ES/slides/TestesDeSoftware.pdf>>. Acesso em: 15 out. 2008.
- LIMA, G. A. F. e ROCHA NETO, A. F. **Sistemas institucionais integrados da UFRN**. Disponível em: <http://www.iiwtiifes.ufrgs.br/trabalhos/TRAB10943_CPE55640_53_ArtigoUFRN.pdf>. Acesso em: 22 set. 2008.
- MACORATTI, J. C. **Padrões de projeto: O Modelo MVC – Model View Controller**. Disponível em: <http://www.macoratti.net/vbn_mvc.htm>. Acesso em: 10 nov. 2008.
- MACORATTI, J. C. **Testes em desenvolvimento de software**. Disponível em: <http://www.macoratti.net/tst_sw1.htm>. Acesso em: 27 ago. 2008
- MOLINARI, Leonardo **Testes funcionais de software**. Florianópolis: Visual Books, 2008.

DIAS NETO, A. C. **Introdução a teste de software**. Engenharia de Software Magazine, Rio de Janeiro, ano 1, n.1, p. 54-59, 2007. Disponível em: <http://kameha.devmedia.com.br/site/engsoft/es_baixa.zip>. Acesso em: 15 out. 2008.

OLIVEIRA, M. **Técnicas de testes caixa-preta**. Disponível em: <http://ebts2007.cesar.org.br/Apresentacoes/Mini-cursos/II-EBTS_TestesCaixaPreta_OUT-2007.ppt#293,21,Tabela de Decisão>. Acesso em: 27 set. 2008.

PEREIRA, N. B. M. e ZANATTA, T. **Artigo sobre MVC**. Disponível em: <http://anacarol.blog.br/old/aulas/artigos_uteis/modelo_visualizacao_controle.pdf>. Acesso em: 11 nov. 2008.

PETERS, James E. e PEDRYCZ, Witold. **Engenharia de software: teoria e prática**. Rio de Janeiro: Campus, 2001.

PRESSMAN, Roger. S. **Engenharia de software**. São Paulo: Makron Books, 1995.

RIBEIRO, R. L. **Uma visão para aplicações orientadas a objetos - II**. Disponível em: <<http://www.mundooo.com.br/php/modules.php?name=MOOArtigos&pa=showpage&pid=14>>. Acesso em: 27 set. 2008.

ROCHA, A. C. O. **Casos de testes fundamentais**. Disponível em: <<http://gtsw.blogspot.com/2008/02/casos-de-teste-fundamentais.html>>. Acesso em: 15 out. 2008.

SAUVÉ, J. P. **Desenvolvimento de aplicações corporativas avançadas**. Disponível em: <<http://jacques.dsc.ufcg.edu.br/cursos/j2ee/html/j2ee-2.htm>>. Acesso em: 10 nov. 2008.

SOLAMO, M. R. **Engenharia de software: Módulo 4 – Lição 6 - Teste de Software**. Java Education & Development Initiative – DFJUG. Disponível em: <<http://jedi.wv.com.br/mod/resource/view.php?id=215>>. Acesso em: 17 out. 2008.

Superintendência de Informática. Disponível em: <<http://www.info.ufrn.br/conteudo/organograma/organograma.html>>. Acesso em: 16 out. 2008.

TELEMACO, U. **Programação de aplicações Web – MVC**. Natal, 2007.

TOZELLI, P. **Teste de software**. Disponível em:

<http://imasters.uol.com.br/artigo/9572/des_de_software/teste_de_software>. Acesso em: 22 set. 2008a.

TOZELLI, P. **Análise do teste de software** – Parte 02. Disponível em:

<http://imasters.uol.com.br/artigo/9817/des_de_software/analise_do_teste_de_software_-_parte_02/>. Acesso em: 20 out. 2008b.

VILLAFUERTE, D. **Programação WEB**: Módulo 6 – Lição 7 – introdução a MVC e ao Framework Struts. Java Education & Development Initiative – DFJUG. Disponível em: <<http://jedi.wv.com.br/mod/resource/view.php?id=291>>. Acesso em: 9 nov. 2008a.

VILLAFUERTE, D. **Programação WEB**: Módulo 6 – Lição 9 – MVC Frameworks II – Java Server Faces. Java Education & Development Initiative – DFJUG. Disponível em: <<http://jedi.wv.com.br/mod/resource/view.php?id=295>>. Acesso em: 28 out. 2008b.